United States Patent [19]

Santon et al.

[11] **Patent Number: 5,058,162**

[45] **Date of Patent: Oct. 15, 1991**

[54] **METHOD OF DISTRIBUTING COMPUTER DATA FILES**

[75] Inventors: **John C. Santon**, Johnstown; **Kenneth R. Nielsen**, Loveland, both of Colo.; **Christen M. Armbrust**, Boulder Creek, Calif.; **Pankaj B. Shah**, Santa Clara, Calif.; **Steven J. Hand**, San Jose, Calif.

[73] Assignee: **Hewlett-Packard Company**, Palo Alto, Calif.

[21] Appl. No.: **564,911**

[22] Filed: **Aug. 9, 1990**

[51] **Int. Cl.**[5] **H04K 1/00**; H04K 9/00; H04L 9/00

[52] **U.S. Cl.** **380/25**; 364/200; 364/900; 380/4

[58] **Field of Search** 364/200, 900; 380/23, 380/25, 45, 4

[56] **References Cited**

U.S. PATENT DOCUMENTS

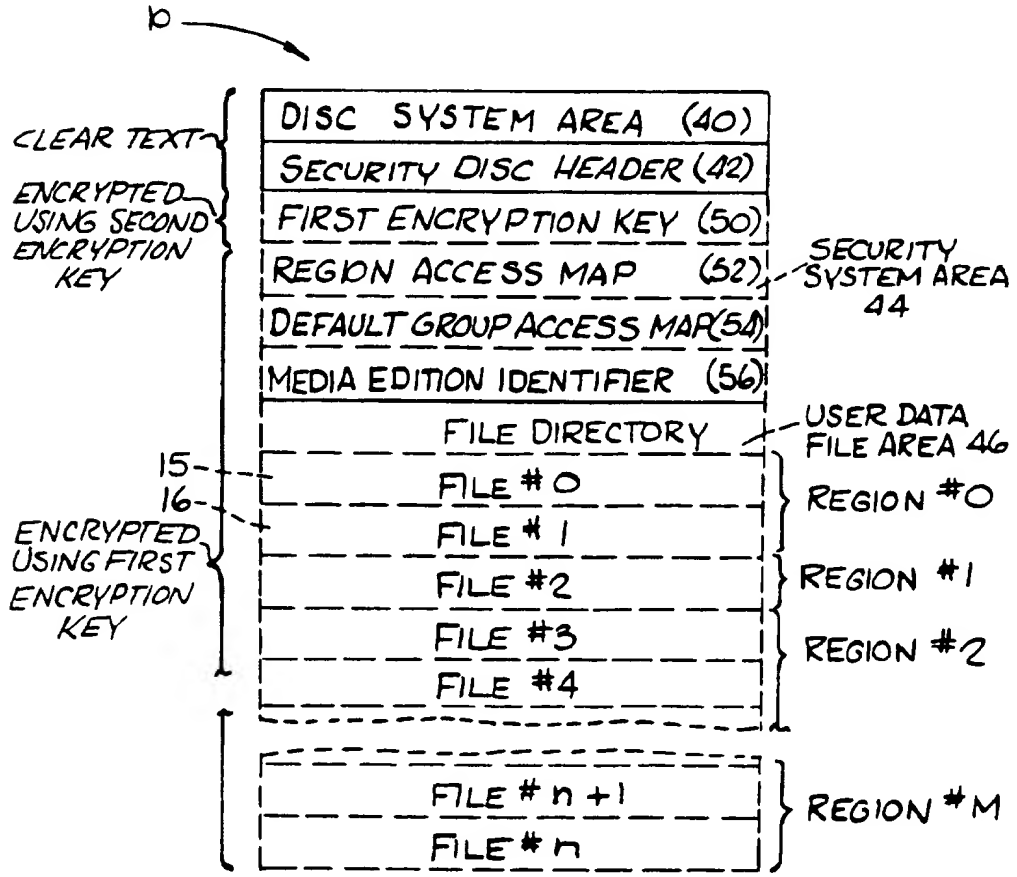
3,670,310	6/1972	Bharwami	235/157
4,604,686	8/1986	Reiter et al.	364/200
4,757,533	7/1988	Allen et al.	380/25

Primary Examiner—Stephen C. Buczinski

[57] **ABSTRACT**

A method of distributing a plurality of data files to a plurality of recipients including the steps of: placing encrypted copies of the data files to be distributed on a plurality of identical media and providing the recipients with media reading devices having data file decryption capability; logically arranging the data files into data file groups; in response to a recipient's request for access to selected file groups providing the recipient with a group access map indicative of the file groups to which access is requested; in further response to a recipient's request for access to selected file groups providing the recipient with a password to be used for access verification; completing an access verification operation using the group access map and the password and data indicative of the media being read and data indicative of the reading device being used; providing access to the data files in the file groups to which access is requested by use of the group access map; and decrypting the accessed data files.

17 Claims, 5 Drawing Sheets



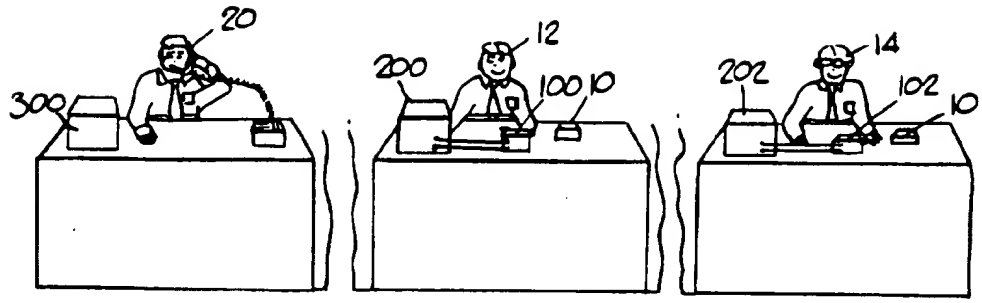

20
300
12
200
100
10
14
202
102
10

FIG. 1

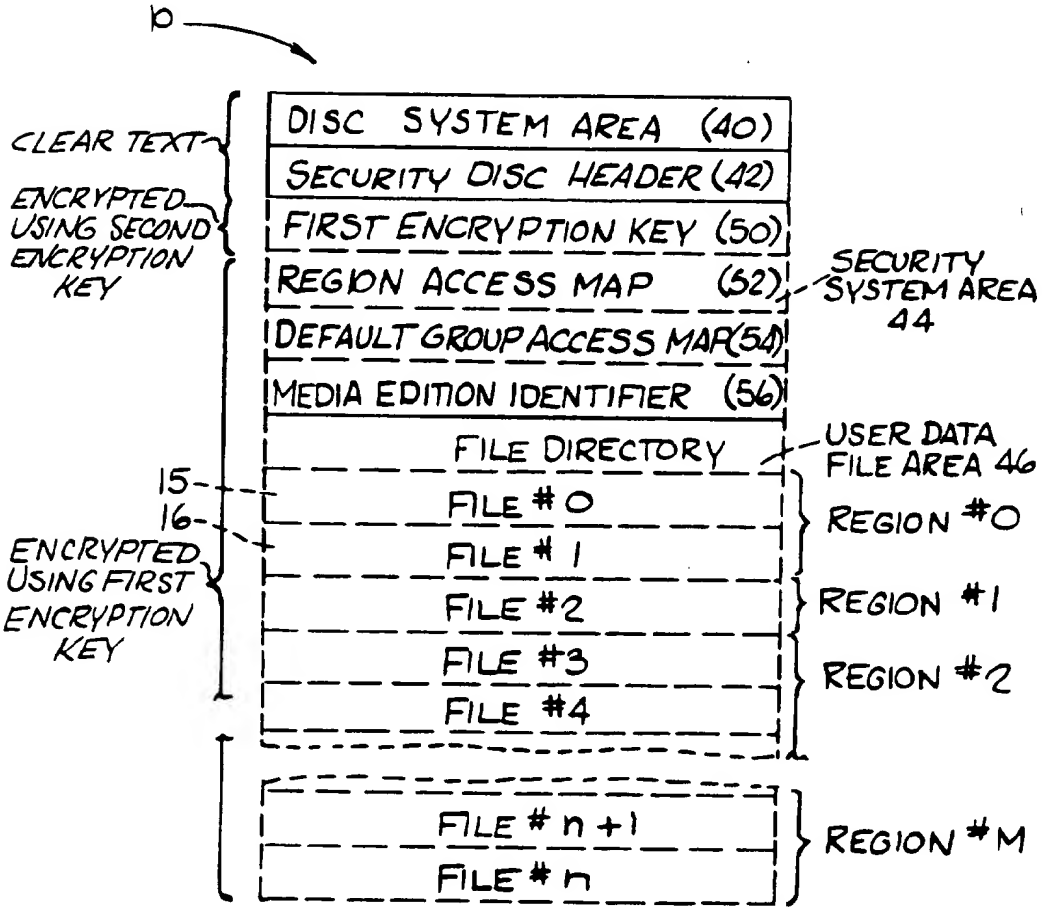

10
CLEAR TEXT
ENCRYPTED USING SECOND ENCRYPTION KEY
ENCRYPTED USING FIRST ENCRYPTION KEY
DISC SYSTEM AREA (40)
SECURITY DISC HEADER (42)
FIRST ENCRYPTION KEY (50)
REGION ACCESS MAP (52)
DEFAULT GROUP ACCESS MAP (54)
MEDIA EDITION IDENTIFIER (56)
SECURITY SYSTEM AREA 44
FILE DIRECTORY
USER DATA FILE AREA 46
15
16
FILE # 0
FILE # 1
FILE #2
FILE #3
FILE #4
FILE # n+1
FILE # n
REGION #0
REGION #1
REGION #2
REGION #M

FIG. 2

REGION STARTING ADDRESS	ASSIGNED GROUP # OF REGION
0	7
26	1
92	24
250	0
312	2
419	0
586	15
635	4
770	5
922	4
2920	18
3356	2

62 64 52

FIG. 3

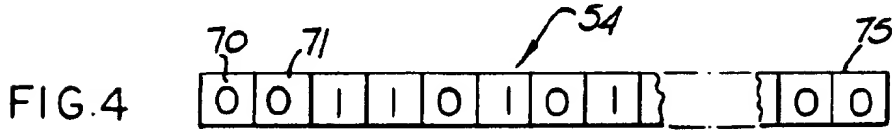


FIG. 4

FIG. 5

100

DRIVE ELECTRO/ MECHANICAL ASSEMBLY (110)	DRIVE CONTROLLER HARDWARE (112)	DRIVE CONTROLLER FIRMWAVE (114)
		SECOND (124) ENCRYPTION KEY
	DECRYPTION CHIP (120)	UNIQUE ORIVE IDENTIFIER (126)
		SECURITY SOFT-WARE PROGRAM (128)

FIG. 6

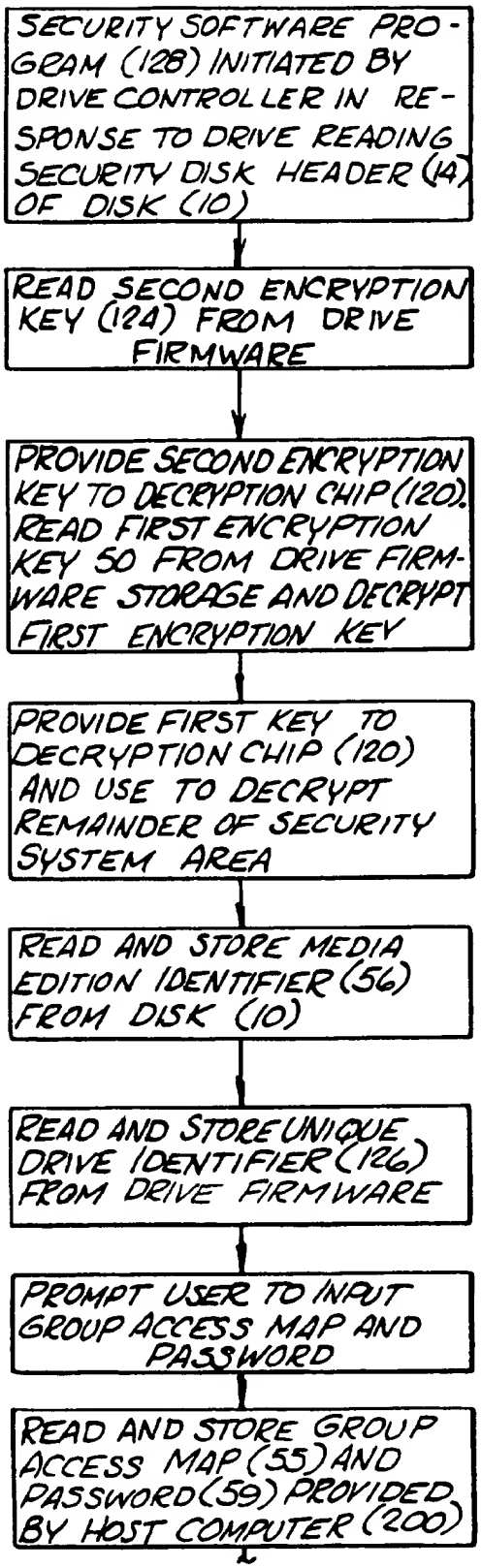

SECURITY SOFTWARE PROGRAM (128) INITIATED BY DRIVE CONTROLLER IN RESPONSE TO DRIVE READING SECURITY DISK HEADER (14) OF DISK (10)
READ SECOND ENCRYPTION KEY (124) FROM DRIVE FIRMWARE
PROVIDE SECOND ENCRYPTION KEY TO DECRYPTION CHIP (120). READ FIRST ENCRYPTION KEY 50 FROM DRIVE FIRMWARE STORAGE AND DECRYPT FIRST ENCRYPTION KEY
PROVIDE FIRST KEY TO DECRYPTION CHIP (120) AND USE TO DECRYPT REMAINDER OF SECURITY SYSTEM AREA
READ AND STORE MEDIA EDITION IDENTIFIER (56) FROM DISK (10)
READ AND STORE UNIQUE DRIVE IDENTIFIER (126) FROM DRIVE FIRMWARE
PROMPT USER TO INPUT GROUP ACCESS MAP AND PASSWORD
READ AND STORE GROUP ACCESS MAP (55) AND PASSWORD (59) PROVIDED BY HOST COMPUTER (200)

FIG.7A

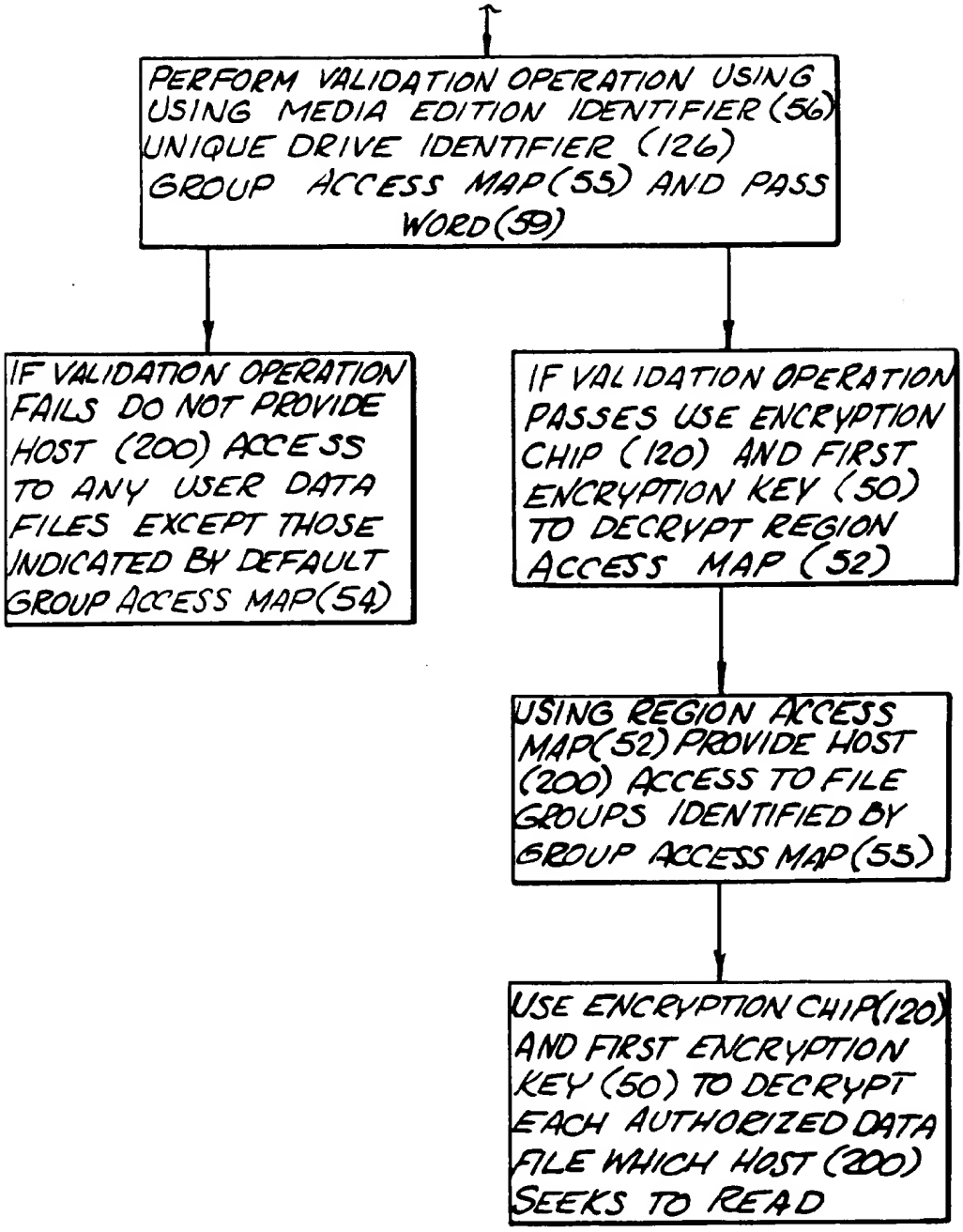

PERFORM VALIDATION OPERATION USING USING MEDIA EDITION IDENTIFIER (56) UNIQUE DRIVE IDENTIFIER (126) GROUP ACCESS MAP (55) AND PASS WORD (59)
IF VALIDATION OPERATION FAILS DO NOT PROVIDE HOST (200) ACCESS TO ANY USER DATA FILES EXCEPT THOSE INDICATED BY DEFAULT GROUP ACCESS MAP (54)
IF VALIDATION OPERATION PASSES USE ENCRYPTION CHIP (120) AND FIRST ENCRYPTION KEY (50) TO DECRYPT REGION ACCESS MAP (52)
USING REGION ACCESS MAP (52) PROVIDE HOST (200) ACCESS TO FILE GROUPS IDENTIFIED BY GROUP ACCESS MAP (55)
USE ENCRYPTION CHIP (120) AND FIRST ENCRYPTION KEY (50) TO DECRYPT EACH AUTHORIZED DATA FILE WHICH HOST (200) SEEKS TO READ

FIG. 7B

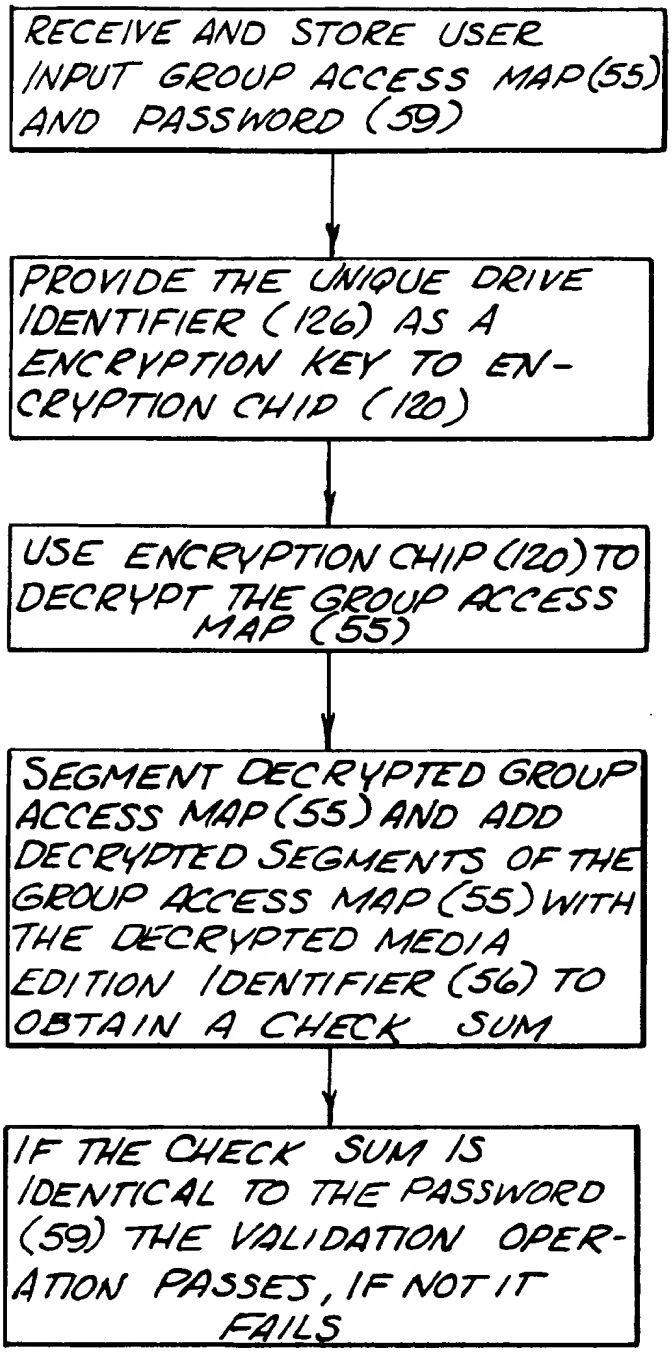

RECEIVE AND STORE USER INPUT GROUP ACCESS MAP (55) AND PASSWORD (59)
PROVIDE THE UNIQUE DRIVE IDENTIFIER (126) AS A ENCRYPTION KEY TO ENCRYPTION CHIP (120)
USE ENCRYPTION CHIP (120) TO DECRYPT THE GROUP ACCESS MAP (55)
SEGMENT DECRYPTED GROUP ACCESS MAP (55) AND ADD DECRYPTED SEGMENTS OF THE GROUP ACCESS MAP (55) WITH THE DECRYPTED MEDIA EDITION IDENTIFIER (56) TO OBTAIN A CHECK SUM
IF THE CHECK SUM IS IDENTICAL TO THE PASSWORD (59) THE VALIDATION OPERATION PASSES, IF NOT IT FAILS

FIG.8

METHOD OF DISTRIBUTING COMPUTER DATA FILES

BACKGROUND OF THE INVENTION

The present invention relates generally to a method of distributing computer software and, more particularly to a method of distributing computer software which enables a software distributor to provide multiple data files on a distribution media, to distribute identical copies of the media to a plurality of potential software users, and to selectively provide different software users with access to different sets of data files contained on the media.

Computer data files, e.g. word processing software, engineering application software, mailing lists, etc., are generally distributed by software publishers to software users on media which contain data files for only that software which the user has purchased. However in the case of institutional software distribution such as distribution to universities or large commercial organizations, publishers often bundle together a large number of different software programs on a single media volume such as a large data storage tape. Identical copies of this large capacity volume are then distributed to each of the publisher's institutional clients. Updated copies of the large capacity volume are generally distributed to clients at periodic intervals, e.g. monthly. A significant cost savings is achieved by the publisher through such large volume distribution of software due to efficiencies in both media costs and copying costs as compared to costs of preparing and distributing individual software programs on separate, low capacity media.

However, a drawback to this method of distributing institutional software on identical large volumes has been that not all institutional customers are interested in the same software. Typically each customer is interested in only some small portion of the software which is available on each large volume. Yet the customer, in order to gain access to the software which is desired, must pay for the entire volume. Due to the practical difficulty in requiring a customer to pay for software which the customer does not desire, the unit price for software which is distributed on such large identical volumes must necessarily be much lower than the unit price of software which is distributed as individual programs on small capacity media. An associated problem is that once a large volume of media is released to a customer, all of the software programs contained on the volume may be subject to unauthorized copying.

The copying costs associated with providing large "customized" software volumes to each institutional customer which contains only the software which each customer has actually ordered make this method of distribution expensive.

The present invention seeks to combine the media cost and copying cost efficiencies associated with providing numerous data files on a single large capacity media volume with the pricing efficiencies associated with providing individual software programs on separate low capacity media volumes. The present invention is particularly adapted to exploit the high capacity and relatively low copy cost associated with publishing software on ROM disks.

SUMMARY OF THE INVENTION

The present invention is directed to a software distribution method which achieves a number of desireable results for a software distributor.

All of the software which is to be distributed to a particular set of customers may be distributed on identical large capacity media volumes rather than media volumes which are unique to each customer. Thus the distributor's production costs are significantly reduced as compared to customer specific distribution methods.

The distributors's customers are provided with specially adapted reading devices. Data files on the media are encrypted and only those software users which have been provided with the specially adapted reading devices are capable of reading the data files in a usable form. The encryption "keys" needed to decrypt the data files are provided on the media and the specially adapted reading devices but are not directly accessible by software users. Thus, the risk of authorized software users making unauthorized use of the encryption keys or divulging the encryption keys to unauthorized users is obviated.

By use of a technique employing two encryption keys, a unique key for each new media release which is provided on the media itself and a "generic" key which is provided on each media reading device, each new edition of the media may be encrypted with a different key and yet still be decryptable by use of the specially adapted reading devices without modification of the firmware of the reading devices.

A technique of providing a customer a "group access map" which is compared by a security program installed in the reading device to a "region access map" on the media allows the software distributor to precisely designate different file group access for different customers.

The invention employs a password based access verification procedure which utilizes identifiers specific to a particular media release, a particular reading device and a particular set of data files. This verification procedure limits a customer's access to only those data files of a particular media release for which he has been granted access authorization by the distributor. The particular password provided to one customer is not usable by other customers due to the system's use of different reading device identifiers. Similarly the password used by a customer to gain access to files on one media release is not usable to gain access to files on a subsequent release due to the use of different media identifiers on each new release.

The present invention may comprise a method of distributing a plurality of data files to a plurality of recipients including the steps of: placing encrypted copies of the data files to be distributed on a plurality of identical media and providing the recipients with media reading devices having data file decryption capability; logically arranging the data files into data file groups; in response to a recipient's request for access to selected file groups providing the recipient with a group access map indicative of the file groups to which access is requested; in further response to a recipient's request for access to selected file groups providing the recipient with a password to be used for access verification; completing an access verification operation using the group-access map and the password and data indicative of the media being read and data indicative of the reading device being used; providing access to the data files in

the file groups to which access is requested by use of the group access map; and decrypting the accessed data files.

The invention may also comprise a method of distributing a plurality of data files to a plurality of recipients including the steps of: creating a plurality of identical media which each contain copies of the plurality of data files encrypted with a first encryption key and which each contain a copy of the first encryption key encrypted with a second encryption key; providing each of the recipients with a media reading device having a machine readable copy of the second encryption key stored therein; initiating reading of one of the media on one of the media reading devices; reading the stored copy of the second encryption key; using the read copy of the second encryption key to decrypt the encrypted copy of the first encryption key which is provided on the media; and using the decrypted first encryption key to decrypt the encrypted data files on the media.

The invention may also comprise a method of distributing a plurality of data files to a plurality of recipients including the steps of: creating a set of identical media which contain copies of the plurality of data files, a media identifier, and a security program initiator; providing the recipients with secured media reading devices which each contain a common security program and which each contain a unique reading device identifier; initiating reading of one of the media on one of the media reading devices; initiating the security program in response to reading the security program initiator; inputting to the reading device a group access map indicative of particular files which are to be accessed; inputting to the reading device a password which is correlated to the unique reading device identifier of the reading device being used and to the media identifier of the media being read and to the input group access map; utilizing the security program to access the unique reading device identifier of the reading devices and the media identifier of the media being read; performing a verification operation to establish that a predetermined correlation exists among the password, reading device identifier, media identifier, and group access map; and providing access to the selected data files indicated by the group access map in response to establishing the correlation.

The invention may also comprise a method of providing access to selected sets of data files which are provided on a digital data storage media including the steps of: logically assigning each data file to a file group based upon predetermined criteria and assigning a unique file group number to each file group; logically dividing the area on the disk on which data files are stored into a plurality of contiguous physical regions wherein each data file is contained within a single region and wherein no region contains data files of more than one file group; providing a region access map indicating the group number of the files in each region and indicating the disk location of each region and providing the region access map list on the media; creating a group access map indicating the file group numbers of the file groups to which access is desired; comparing the group access map to the region access map to determine the region locations of the data files in each of the file groups to which access is desired; and providing access to each of these determined regions.

The invention may also comprise a method of distributing a plurality of data files to a plurality of recipients including the steps of: creating a plurality of identical media which each contain copies of the plurality of data files encrypted with a first encryption key, a copy of the first encryption key encrypted with a second encryption key, a media identifier, and a security program initiator; logically assigning each data file to a file group based upon predetermined criteria and assigning a unique file group number to each file group; logically dividing the area on the disk on which data files are stored into a plurality of contiguous physical regions wherein each data file is contained within a single region and wherein no region contains data files of more than one file group; providing a region access map indicating the group number of the files in each region and indicating the disk location of each region and providing the region access map on the media; creating a group access map indicating the file group numbers of the file groups to which access is desired; providing each of the recipients with a media reading device having firmware including a copy of the second encryption key, a unique reading device identifier, and a security program; initiating reading of one of the media on one of the media reading devices; initiating the security program in response to reading the security program initiator; inputting to the reading device the group access map indicative of particular file groups which are to be accessed; inputting to the reading device a password which is correlated to the unique reading device identifier of the reading device being used and to the media identifier of the media being read and to the input group access map; utilizing the security program to access the unique reading device identifier of the reading devices and the media identifier of the media being read; utilizing the security program to perform a verification operation to establish that a predetermined correlation exists among the password, reading device identifier, media identifier, and group access map; utilizing the security program to compare the group access map to the region access map to determine the region locations of the data files in each of the file groups to which access is desired and providing a recipient access to each of these determined regions; utilizing the security program to read the stored copy of the second encryption key; utilizing the security program to use the read copy of the second encryption key to decrypt the encrypted copy of the first encryption key which is provided on the media; and utilizing the security program to use the decrypted first encryption key to decrypt the encrypted data files on the media regions accessed by the recipient.

BRIEF DESCRIPTION OF THE DRAWING

An illustrative and presently preferred embodiment of the invention is shown in the accompanying drawings in which:

FIG. 1 is a schematic illustration of a software distributor, software users, software distribution media, and media reading devices.

FIG. 2 is a schematic illustration of the contents of a secured software distribution media.

FIG. 3 is an illustration of typical data contained in a region access map.

FIG. 4 is an illustration of typical data contained in a default group access map.

FIG. 5 is an illustration of typical data contained in a group access map and password.

FIG. 6 is a schematic drawing of major components of a disk drive.

FIG. 7 is a block diagram illustrating the operation of a security software program.

FIG. 8 is a block diagram illustrating a group access map validation operation of a security software program.

DETAILED DESCRIPTION OF THE INVENTION

As illustrated in FIG. 1, identical copies of a secured media such as read-only, compact laser disk 10 (hereinafter ROM disk 10 or simply disk 10) are supplied to a plurality of different software users 12, 14 by a software distributor 20. Each disk 10 contains many different data files 15, 16, etc., FIG. 2. Each data file belongs to a particular file group. Each of these file groups provides a software module which a software user may desire to use. For example, one file group may contain a single word processing program; another file group may contain a structural engineering design program and an engineering drawing program; another file group may contain a text file telephone directory of a particular geographic region, etc.

The disks 10 are secured in a manner which enables the distributor to limit access to data files on the disk to software users having media reading devices, e.g. 100/200, which have been provided with special security identifier data and software by the distributor. The distributor is able to limit the access of software users having such special reading devices to only those file groups to which a particular software user has been granted access authority by the distributor.

Each ROM disk 10 has a continuous spiral-shaped path upon which data is stored. FIG. 2 is a schematic illustration showing different areas of the disk on which information is stored. The first area on which information is stored corresponds to the area which the drive 100 would read first at system start-up. This area is a conventional disk system area 40 and contains data which enables the disk reader to properly interface with the disk being read. Such start-up regions on ROM disks are conventional and well-known in the art. The disk system area 40 is written in clear text, i.e. the data is not encrypted and may be read by any ROM disk reading device.

A security disk header 42 is provided in the area of the disk immediately after the disk system area 40. The security disk header 42 is also written in clear text and provides data which alerts the reading device to the fact that the disk being read is a secured disk.

The next area immediately after the security disk header 42 is the security system area 44 which contains various data used in implementing disk security.

The first region in security system area 44 contains a first encryption key 50. The security system area 44 also contains a region access map 52, a default group access map 54, and a media edition identifier 56, each of which is described in further detail below.

The next disk area immediately after the security system area 44 is the user data file area 46 which contains a file directory and a series of data files 15, 16, etc., which are numbered sequentially in the drawing for purposes of explanation as file no. "0" through file no. "n". Each file group which a user may desire to obtain access to includes one or more of these data files. However the data files which belong to any particular file group are not necessarily contiguous.

One technique which is used according to the present invention for securing media 10 is "data encryption". Data encryption refers to the scrambling of a set of data according to a set procedure such that the scrambled data is unusable by unauthorized parties but such that the data may be unscrambled by parties having knowledge of the scrambling procedure. One method of encrypting data known in the industry as the "Data Encryption Standard" or "DES" utilizes a device such as a microprocessor which employs a plurality of algorithms which operate on an input data string and which output a scrambled string of data of identical size to that which was input. The algorithms which operate on the input data may be varied in accordance with a predetermined procedure which is determined by a separate set of data referred to as an "encryption key" or simply a "key". The DES device may be operated in a decryption mode to perform an unscrambling operation on decrypted data which is the inverse of the encryption operation. Thus, through use of a DES chip operating in an encryption mode which is supplied with a predetermined encryption key, a set of data files may be encrypted in a manner which renders them unusable by unauthorized personnel. These encrypted data files may subsequently be returned to their original form through use of an identical encryption chip operating in a decryption mode which has been supplied with the same encryption key which was used to encrypt the files. The media security system of the present invention employs data encryption according to a novel method.

The first encryption key 50 which is stored in the first region of the security area is encrypted using a predetermined encryption standard such as that provided by using predetermined encryption software, or, in the preferred embodiment, by using an encryption chip which may be a Data Encryption Standard (DES) chip such as a Z8068 Data Cyphering Processor manufactured by Zilog, Inc., 210 Hacienda Avenue, Campbell, Calif. 95008-6609. The encryption of the first encryption key is performed using a second encryption key with the encryption device. The remainder of the system area 44 and the user data file area 46 is encrypted using the same encryption device and using the first encryption key (in its unencrypted form) with the encryption chip.

The region access map 52 is a security directory, FIG. 3, which logically divides the user data file area 46 of the disk into a series of contiguous physical regions for the purpose of identifying where data files assigned to various file groups are located. Each defined region in the user data file area 46 contains one or more contiguous files of a single file group but not necessarily all of the files of that particular file group. In other words, each region is assigned to a group. Multiple regions can be assigned to a single group, but a single region can only be assigned to one group. The region access map comprises a listing 62 of the starting address of each region and a corresponding list 64 of the group to which each region is assigned. Each region terminates at the beginning of the next succeeding region. FIG. 2 schematically illustrates how the user data file area is divided into contiguous regions which are numbered "0" through "m" for purposes of explanation.

FIG. 4 illustrates the default group access map 54 which simply comprises a string of "p" binary digits. The first digit in the string, indicated by reference numeral 70, represents the security status of the first file group, the second binary digit 71 represents the security status of the second file group, the pth digit in the string 75 represents the security status of the pth file group on

the disk. In one preferred embodiment, a security status of unlocked is represented with a "0", and the security status of locked is represented with a "1". The default group access map **54** is used by security software **128** in association with the region access map **52** to allow or prevent access to the various file groups on the disk, as further explained below. The default group access map designates a "default set" of file groups to which a user is provided access without entering a password. A user may obtain access to additional file groups by inputting a new group access map **55** and a corresponding password **59** supplied by the software distributor. The new group access map **55** which is input by the customer is identical in form to the default group access map **54** but designates different file group access authority. In one embodiment of the invention the new group access map **54** and the corresponding password **59** are provided to the software user by the distributor as a single codeword **55/59** as illustrated in FIG. **5**.

As previously mentioned, each user is provided with a reading device which may comprise a disk drive **100** and a host computer **200**. The reading device **100/200** is configured by the distributor to implement the disk security function when a user attempts to read a secured disk **10** and otherwise operates in the same manner as a conventional disk reading device. In one preferred embodiment, the various security features of the reading device are all provided on the disk drive **100**, as opposed to the host computer **200**. Various components of the disk drive **100** are illustrated in FIG. **6**. The disk drive may comprise a conventional drive electromechanical assembly **110**. The drive is also provided with conventional drive controller hardware **112** and drive controller firmware **114**. In addition to the conventional controller hardware, the drive is provided with a decryption chip **120** which in one preferred embodiment is a Data Encryption Standard (DES) chip such as a Z8068 Data Cyphering Processor manufactured by Zilog, Inc., 210 Hacienda Avenue, Campbell, Calif. 95008-6609. The decryption chip **120** may comprise an identical chip to that which is used to encrypt the first encryption key **50** and user data files **15**, **16** which are provided on the ROM disk **10**. However, the chip on the drive controller is used only in the decryption mode of operation.

The drive controller firmware **114**, in addition to conventional firmware, comprises a second encryption key **124**, a unique drive identifier **126**, and a security software program **128**.

The method by which a software user obtains access to a data file group will now be described at the software user level.

The software user **12** is provided with a ROM disk **10** and a disk drive **100** which has been configured by the software distributor **20** as described above. The disk drive in addition to novel security features may contain conventional ROM disk drive and magnetic disk drive features such as those provided on Models 9127A and 9135C magnetic disk drives manufactured and sold by Hewlett-Packard Company. The disk drive **100** is operably attached to the user's computer **200** which may comprise a conventional minicomputer or microcomputer such as a Series 300 minicomputer manufactured and sold by Hewlett-Packard Company.

The software user **12** is also provided by the software distributor **20** with a list of data file groups to which he may obtain access by inputting the proper code number. The software user selects one or more file groups from this list and then contact the software distributor **20** and informs the distributor of his selection. The distributor has in his possession, such as in a computer storage device **300**, a list indicating the unique drive identifier for the drive **100**, **102**, etc. which is assigned to each software user **12**, **14**, etc. Based upon the unique drive identifier assigned to the drive of the software user **12** requesting access, and based upon the file groups which the software user **12** desires to access, and based upon the media edition identifier of the ROM disk **10** which the software user **12** has in his possession, the distributor determines a unique code word **55/59**, FIG. **5**, which he furnishes to the software user **12**. The software user **12** then places disk **10** in the disk drive and initiates operation of the disk drive. He next inputs the code word **55/59** provided to him by the distributor when prompted by his host computer **200**. If the code word provided to him by the distributor **20** corresponds correctly to the disk drive **100** and to the edition of the media **10** which the software user **12** is reading, security software on the disk drive **100** provides the user access to the selected file groups and decrypts the data in each of the selected file groups before transmitting it to the host computer **200**.

The various substantive tasks which the security software program **128** of one preferred embodiment performs will now be described with reference to FIG. **7**. The security software program **128** is initiated by other drive controller firmware **114** in response to the drive's reading of the security disk header **14** of disk **10**. Next, the security software program reads the second encryption key **124** from the drive firmware. The security software program next provides the second encryption key to the decryption chip **120**. Next, the security software program reads the encrypted first encryption key **50** from the drive firmware and instructs the decryption chip **120** to decrypt the first encryption key.

The security software program provides the decrypted first encryption key to the decryption chip **120** and instructs the decryption chip **120** to decrypt the media edition identifier from the security system area. Next, the security software program reads and stores the decrypted media edition identifier **56**.

Next, the security software program reads and stores the unique drive identifier **126** from the drive firmware.

Next, the host computer **200** prompts the user to input the code word provided to him by the software distributor **20**. The code word which is input comprises two parts, a group access map **55** and a password **59**, which have different functions within the security program as described in further detail below.

Next, the security software program reads and stores the input group access map and password provided from the host computer **200**.

Next, the security software program performs a validation operation using the media edition identifier **56**, the unique drive identifier **126** and the group access map and the password provided by the user. If the validation operation fails, the user is provided access to only those user data files indicated by the default group access map. If the validation operation passes, the security program provides the encryption chip with the first encryption key **50** and instructs it to decrypt the region access map in the security area of disk **10**.

Next the security program compares the input group access map to the decrypted region access map and unlocks (provides access to) those regions on the disk which are authorized by the group access map.

Finally the security program instructs the encryption chip **120**, using the first encryption key **50** to decrypt any file which the user seeks to access in the unlocked regions of the disk **10**. The data provided to the host computer **200** is clear text.

The validation operation performed by the security software program **128** will now be described in further detail with reference to FIG. **8**. First, the security software provides a copy of the unique drive identifier **126** to the encryption chip **120** which uses it as a key. Next, the security software instructs the chip to decrypt the media edition identifier **56**. Next, the security software instructs the chip **120** to decrypt the group access map portion of the code word which was input by the software user. Next, the security software segments the decrypted group access map into a series of numbers of a predetermined digit length numbers and then adds the individual numbers provided by these segments together. To that sum the security program adds the number represented by the decrypted media edition identifier to obtain a final sum. Next, this final sum or "check sum" is compared to the password **59** which is also represented as a number. If the check sum is identical to the password, then the validation operation passes; if not, the validation operation fails.

It will be appreciated by those having skill in the art that the same method described in FIG. **8** for performing a validation operation may also be performed, except for the last step, by the distributor **20** in order to initially compute the password which must be provided to the software user.

While an illustrative and presently preferred embodiment of the invention has been described in detail herein, it is to be understood that the inventive concepts may be otherwise variously embodied and employed and that the appended claims are intended to be construed to include such variations except insofar as limited by the prior art.

What is claimed is:

1. A method of distributing a plurality of data files to a plurality of recipients comprising the steps of:

a) placing encrypted copies of the data files to be distributed on a plurality of identical media and providing the recipients with media reading devices having data file decryption capability;

b) logically arranging the data files into data file groups;

c) in response to a recipient's request for access to selected file groups providing the recipient with a group access map indicative of the file groups to which access is requested;

d) in further response to a recipient's request for access to selected file groups providing the recipient with a password to be used for access verification;

e) completing an access verification operation using the group access map and the password and data indicative of the media being read and data indicative of the reading device being used;

f) providing access to the data files in the file groups to which access is requested by use of the group access map; and

g) decrypting the accessed data files.

2. The method of claim **1** wherein step (a) comprises:

creating a plurality of identical media which each contain copies of the plurality of data files encrypted with a first encryption key and which each contain a copy of the first encryption key encrypted with a second encryption key;

and further comprising the steps of:

providing each of the recipients with a media reading device having a machine readable copy of the second encryption key stored therein;

initiating reading one of the media on one of the media reading devices;

reading the stored copy of the second encryption key;

and wherein step (e) comprises the steps of:

using the read copy of the second encryption key to decrypt the encrypted copy of the first encryption key which is provided on the media; and

using the decrypted first encryption key to decrypt the encrypted data files on the media.

3. The method of claim **1** comprising the further steps of:

providing a media identifier and a security program initiator on each of the media;

providing a common security program and a unique reading device identifier on each reading device;

initiating reading of one of the media on one of the media reading devices;

initiating the security program in response to reading the security program initiator;

utilizing the security program to access the unique reading device identifier of the reading devices and the media identifier of the media being read;

inputting to the reading device the group access map indicative of particular files which are to be accessed;

inputting to the reading device the password;

and wherein step (g) comprises:

performing a verification operation to establish that a predetermined correlation exists among the password, reading device identifier, media identifier, and group access map.

4. The method of claim **1** comprising the steps of:

logically assigning each data file to a file group based upon predetermined criteria and assigning a unique file group number to each file group;

logically dividing the area on the disk on which data files are stored into a plurality contiguous physical regions wherein each data file is contained within a single region and wherein no region contains data files of more than one file group;

creating a region access map indicating the group number of the files in each region and indicating the disk location of each region and providing the region access map on the media;

and wherein step (f) comprises:

comparing the group access map to the region access map to determine the region locations of the data files in each of the file groups to which access is desired; and

providing access to these determined regions.

5. A method of distributing a plurality of data files to a plurality of recipients comprising the steps of:

creating a plurality of identical media which each contain copies of the plurality of data files encrypted with a first encryption key and which each contain a copy of the first encryption key encrypted with a second encryption key;

providing each of the recipients with a media reading device having a machine readable copy of the second encryption key stored therein;

initiating reading of one of the media on one of the media reading devices;

reading the stored copy of the second encryption key;
using the read copy of the second encryption key to decrypt the encrypted copy of the first encryption key which is provided on the media; and
using the decrypted first encryption key to decrypt the encrypted data files on the media.

6. A method of distributing a plurality of data files to a plurality of recipients comprising the steps of:
creating a set of identical media which contain copies of the plurality of data files, a media identifier, and a security program initiator;
providing the recipients with secured media reading devices which each contain a common security program and which each contain a unique reading device identifier;
initiating reading of one of the media on one of the media reading devices;
initiating the security program in response to reading the security program initiator;
inputting to the reading device a group access map indicative of particular files which are to be accessed;
inputting to the reading device a password which is correlated to the unique reading device identifier of the reading device being used and to the media identifier of the media being read and to the input group access map;
utilizing the security program to access the unique reading device identifier of the reading device and the media identifier of the media being read;
performing a verification operation to establish that a predetermined correlation exists among the password, reading device identifier, media identifier, and group access map; and
providing access to the selected data files indicated by the group access map in response to establishing the correlation.

7. A method of providing access to selected sets of data files which are provided on a digital data storage media comprising the steps of:
logically assigning each data file to a file group based upon predetermined criteria and assigning a unique file group number to each file group;
logically dividing the area on the disk on which data files are stored into a plurality of contiguous physical regions wherein each data file is contained within a single region and wherein no region contains data files of more than one file group;
providing a region access map indicating the group number of the files in each region and indicating the disk location of each region and providing the region access map list on the media;
creating a group access map indicating the file group numbers of the file groups to which access is desired;
comparing the group access map to the region access map to determine the region locations of the data files in each of the file groups to which access is desired; and
providing access to each of these determined regions.

8. A method of distributing a plurality of data files to a plurality of recipients comprising the steps of:
creating a plurality of identical media which each contain copies of the plurality of data files encrypted with a first encryption key, a copy of the first encryption key encrypted with a second encryption key, a media identifier, and a security program initiator;
logically assigning each data file to a file group based upon predetermined criteria and assigning a unique file group number to each file group;
logically dividing the area on the disk on which data files are stored into a plurality of contiguous physical regions wherein each data file is contained within a single region and wherein no region contains data files of more than one file group;
providing a region access map indicating the group number of the files in each region and indicating the disk location of each region and providing the region access map on the media;
creating a group access map indicating the file group numbers of the file groups to which access is desired;
providing each of the recipients with a media reading device having firmware including a copy of the second encryption key, a unique reading device identifier, and a security program;
initiating reading of one of the media on one of the media reading devices;
initiating the security program in response to reading the security program initiator;
inputting to the reading device the group access map indicative of particular file groups which are to be accessed;
inputting to the reading device a password which is correlated to the unique reading device identifier of the reading device being used and to the media identifier of the media being read and to the input group access map;
utilizing the security program to access the unique reading device identifier of the reading device and the media identifier of the media being read;
utilizing the security program to perform a verification operation to establish that a predetermined correlation exists among the password, reading device identifier, media identifier, and group access map;
utilizing the security program to compare the group access map to the region access map to determine the region locations of the data files in each of the file groups to which access is desired and providing a recipient access to each of these determined regions;
utilizing the security program to read the stored copy of the second encryption key;
utilizing the security program to use the read copy of the second encryption key to decrypt the encrypted copy of the first encryption key which is provided on the media; and
utilizing the security program to use the decrypted first encryption key to decrypt the encrypted data files on the media regions accessed by the recipient.

9. The method of claim 2 wherein each media reading device is provided with a recipient inaccessible decryption unit and wherein each decryption step comprises the steps of:
providing a key to the decryption unit; and
using the decryption unit to decrypt data.

10. The method of claim 5 wherein each media reading device is provided with a recipient inaccessible decryption unit and wherein each decryption step comprises the steps of:
providing a key to the decryption unit; and
using the decryption unit to decrypt data.

11. The method of claim 8 wherein each media reading device is provided with a recipient inaccessible

decryption unit and wherein each decryption step comprises the steps of:

providing a key to the decryption unit; and

using the decryption unit to decrypt data.

12. The method of claim 3 wherein the step of performing a verification operation comprises the steps of:

decrypting the group access map using the unique reading device identifier as a key;

performing a mathematical operation involving the decrypted group access map and the media identifier and obtaining a mathematical operation result;

comparing the mathematical operation result to the password; and

providing the recipient access to the file groups indicated by the group access map when the mathematical operation result matches the password.

13. The method of claim 12 wherein the decrypted group access map comprises a string of binary digits of predetermined length and wherein the mathematical operation comprises the steps of:

dividing the decrypted group access map string of binary digits into a predetermined number of equal length segments; and

adding the equal length segments and the media identifier together to obtain a sum.

14. The method of claim 6 wherein the step of performing a verification operation comprises the steps of:

decrypting the group access map using the unique reading device identifier as a key;

performing a mathematical operation involving the decrypted group access map and the media identifier and obtaining a mathematical operation result;

comparing the mathematical operation result to the password; and

providing the recipient access to the file groups indicated by the group access map when the mathematical operation result matches the password.

15. The method of claim 14 wherein the decrypted group access map comprises a string of binary digits of predetermined length and wherein the mathematical operation comprises the steps of:

dividing the decrypted group access map string of binary digits into a predetermined number of equal length segments; and

adding the equal length segments and the media identifier together to obtain a sum.

16. The method of claim 8 wherein the step of performing a verification operation comprises the steps of:

decrypting the group access map using the unique reading device identifier as a key;

performing a mathematical operation involving the decrypted group access map and the media identifier and obtaining a mathematical operation result;

comparing the mathematical operation result to the password; and

providing the recipient access to the file groups indicated by the group access map when the mathematical operation result matches the password.

17. The method of claim 16 wherein the decrypted group access map comprises a string of binary digits of predetermined length and wherein the mathematical operation comprises the steps of:

dividing the decrypted group access map string of binary digits into a predetermined number of equal length segments; and

adding the equal length segments and the media identifier together to obtain a sum.

* * * * *

US005982899A

United States Patent [19]

Probst

[11] Patent Number: 5,982,899

[45] Date of Patent: *Nov. 9, 1999

[54] **METHOD FOR VERIFYING THE CONFIGURATION THE COMPUTER SYSTEM**

[75] Inventor: **Jürgen Probst**, Wildberg, Germany

[73] Assignee: **International Business Machines Corporation**, Armonk, N.Y.

[*] Notice: This patent issued on a continued prosecution application filed under 37 CFR 1.53(d), and is subject to the twenty year patent term provisions of 35 U.S.C. 154(a)(2).

[21] Appl. No.: **08/750,764**

[22] PCT Filed: **Aug. 11, 1995**

[86] PCT No.: **PCT/EP95/03186**

§ 371 Date: **Dec. 16, 1996**

§ 102(e) Date: **Dec. 16, 1996**

[87] PCT Pub. No.: **WO97/07463**

PCT Pub. Date: **Feb. 27, 1997**

[51] **Int. Cl.**[6] **H04L 9/00**; H04L 9/30

[52] **U.S. Cl.** **380/25**; 380/4; 380/9; 380/23; 380/30; 380/49; 380/50

[58] **Field of Search** 380/4, 9, 23, 25, 380/28, 30, 49, 50, 59; 705/410

[56] **References Cited**

U.S. PATENT DOCUMENTS

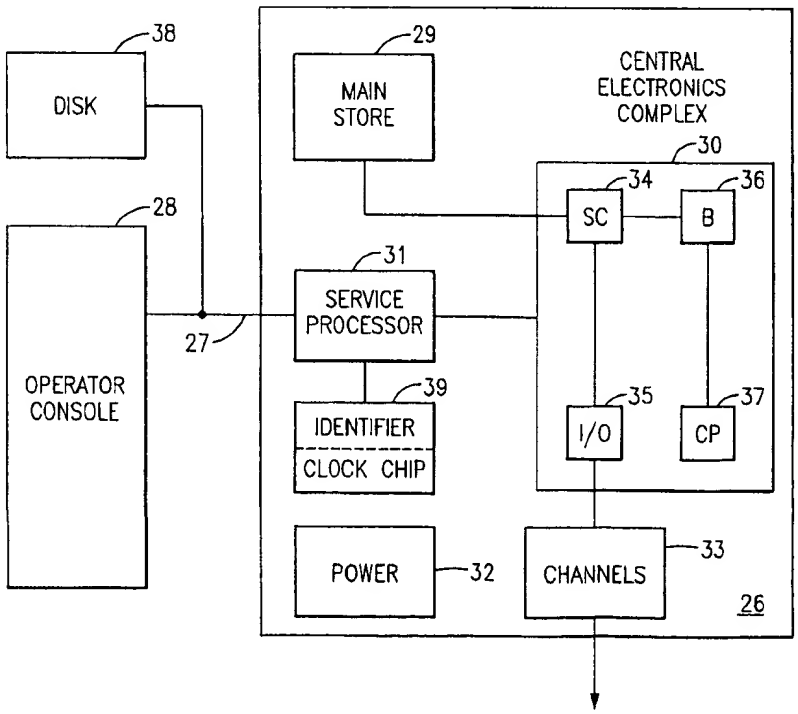
4,796,220	1/1989	Wolfe	380/4
5,077,660	12/1991	Haines et al.	705/410
5,182,770	1/1993	Medveczky et al.	380/4
5,365,587	11/1994	Campbell et al.	380/25
5,388,157	2/1995	Austin	380/4
5,499,295	3/1996	Cooper	380/23
5,553,144	9/1996	Almquist et al.	380/25
5,671,281	9/1997	Campbell et al.	380/25
5,757,907	5/1998	Cooper et al.	380/4

Primary Examiner—Bernarr E. Gregory
Attorney, Agent, or Firm—Marc A. Ehrlich

[57] **ABSTRACT**

A method for verification of configuration data which is expressive of the configuration of a computer system. A computer system having configuration data stored therein, further includes an identifier for uniquely identifying the computer system. A copy of the stored configuration data is encoded via an encoding method which uses the identifier, and the encoded configuration data is encrypted via an encryption method which uses a private key. Subsequently, the encrypted configuration data is decrypted via a decryption method using a public key producing a decrypted result. The decrypted result may either be decoded using the identifier and compared to the stored configuration data or alternatively the stored configuration data may be encoded using the identifier and compared to the decrypted result.

20 Claims, 2 Drawing Sheets



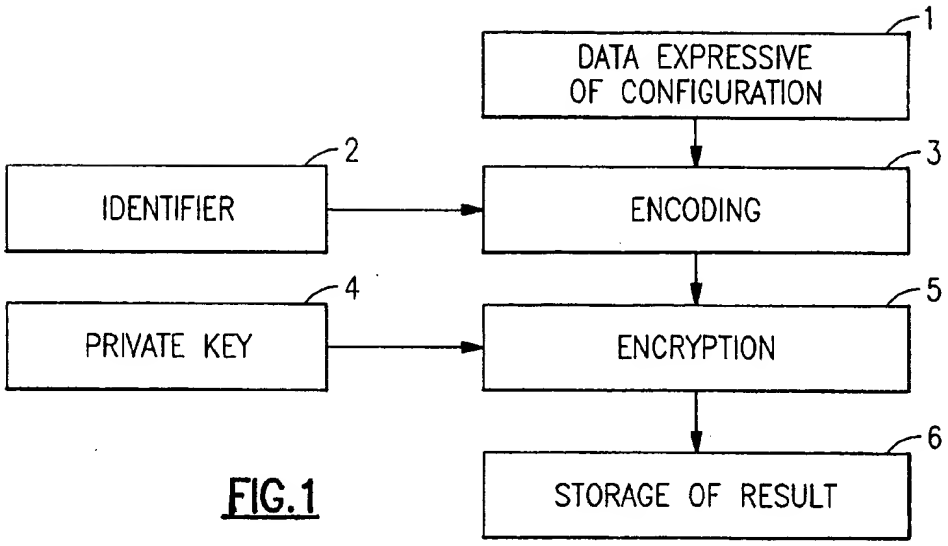

1
DATA EXPRESSIVE OF CONFIGURATION
2
IDENTIFIER
3
ENCODING
4
PRIVATE KEY
5
ENCRYPTION
6
STORAGE OF RESULT

FIG.1

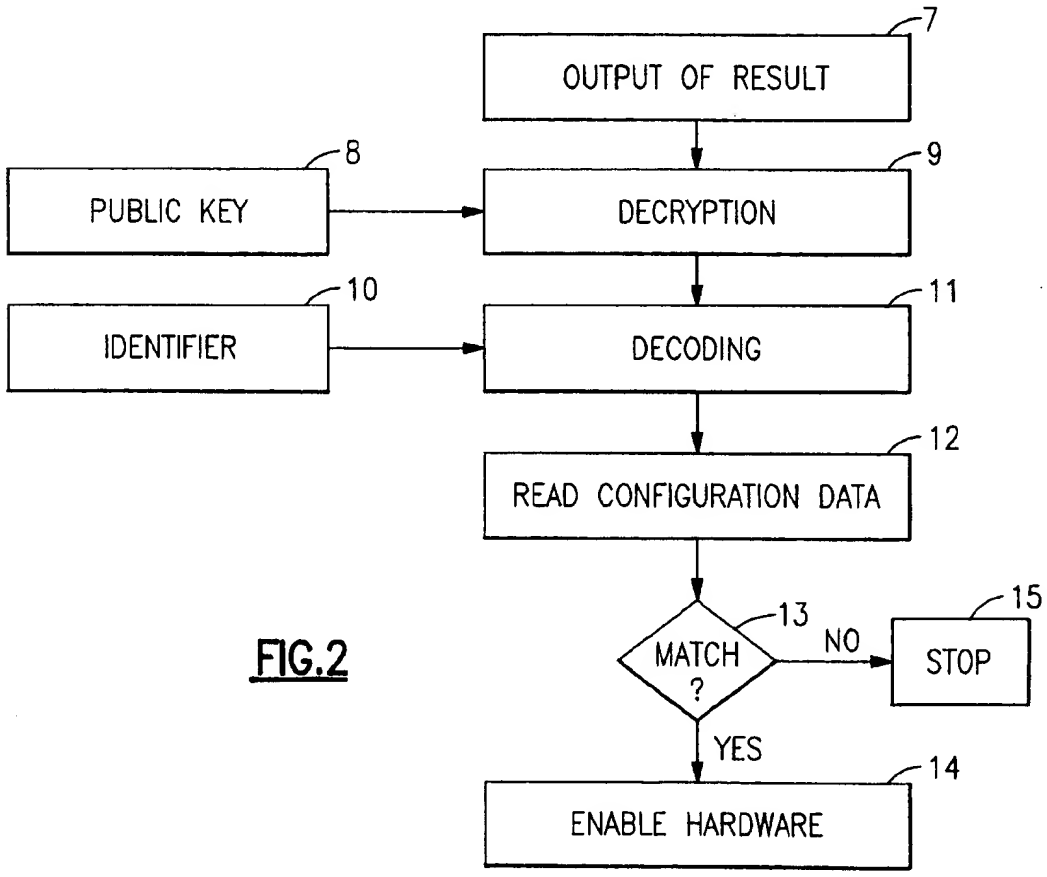

7
OUTPUT OF RESULT
8
PUBLIC KEY
9
DECRYPTION
10
IDENTIFIER
11
DECODING
12
READ CONFIGURATION DATA
13
MATCH ?
NO
15
STOP
YES
14
ENABLE HARDWARE

FIG.2

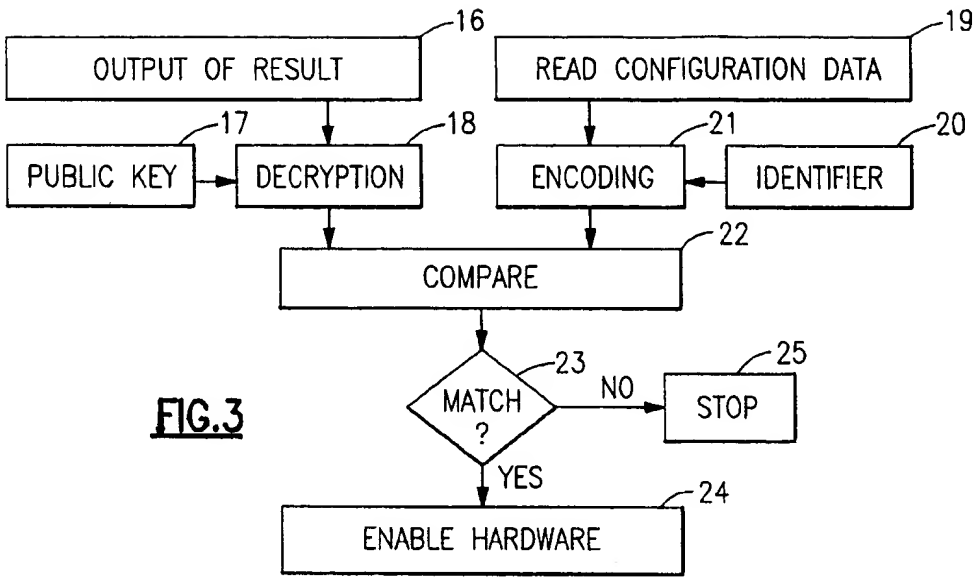

16
OUTPUT OF RESULT
19
READ CONFIGURATION DATA
17
PUBLIC KEY
18
DECRYPTION
21
ENCODING
20
IDENTIFIER
22
COMPARE
23
MATCH ?
NO
25
STOP
YES
24
ENABLE HARDWARE

FIG.3

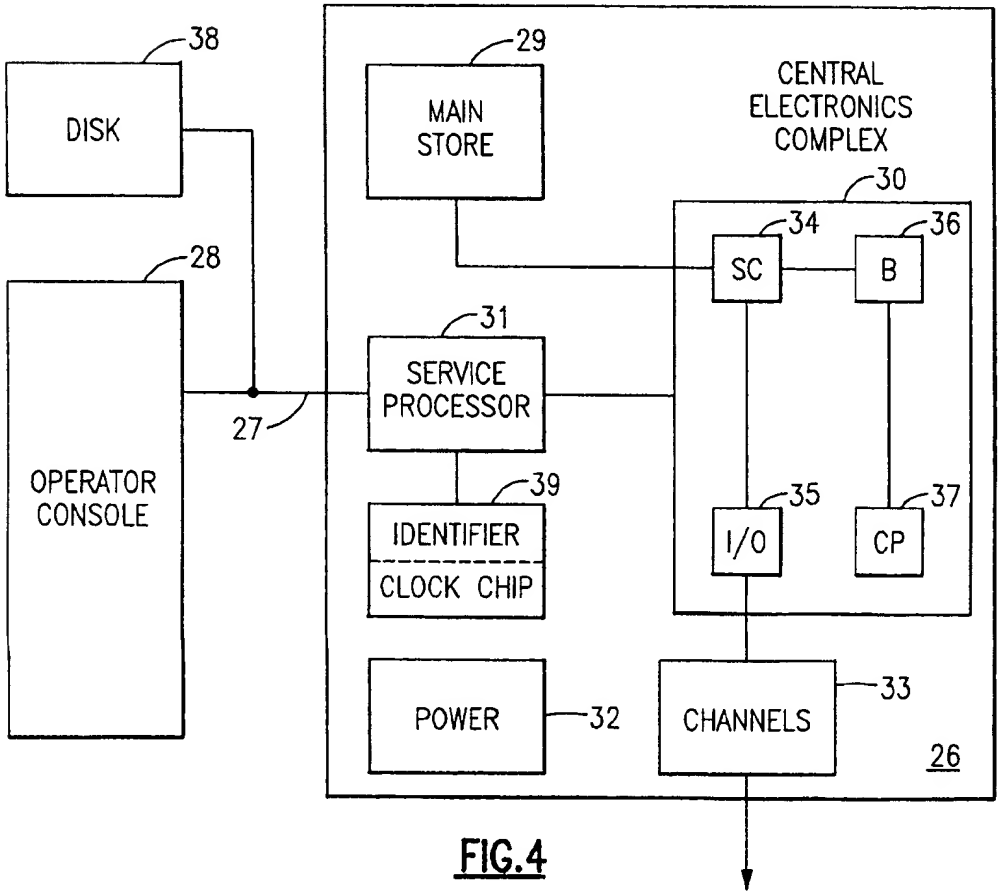

38
DISK
29
MAIN STORE
CENTRAL ELECTRONICS COMPLEX
30
34
SC
36
B
28
31
SERVICE PROCESSOR
27
OPERATOR CONSOLE
39
IDENTIFIER
CLOCK CHIP
35
I/O
37
CP
POWER
32
CHANNELS
33
26

FIG.4

METHOD FOR VERIFYING THE CONFIGURATION THE COMPUTER SYSTEM

METHOD FOR VERIFYING THE CONFIGURATION COMPUTER SYSTEM

The invention relates to a method for verifying configuration of a computer system, to a method for encryption of data being expressive of the configuration of a computer system and to a computer system for carrying out such a method.

It is well known from the prior art to use passwords for security and verification purposes. There have been for many years password generators for a Person Identification Number (PIN). PINs have been used to gain access to automated tellers and security areas when unattended operation and/or verification of authorization is desired. They have been used for granting access to computers, as illustrated by U.S. Pat. No. 4,799,258 to Davies et al granted Jan. 17, 1989. PINS may be generated automatically, and may be generated by random number generators or pseudo-random number sequences stored in the memory of a computer. U.S. Pat. No. 4,800,590 to James C. Vaughan illustrates a password generating device for generating passwords, and a computer access system based upon the generated secure number based on time such that the algorithm is valid only over a 3 minute window. However, the lock or unlock of a computer system e.g. the host computer of Vaughan does not satisfy needs which are now possible to achieve. In particular doesn't deal with repetitive modification of machine function and permits a range of numbers that can be matches instead of one unique number.

Many methods exist for granting or revoking a user's access to selected facilities or files within a data processing system. These techniques often utilize a secret "key" or "password" entered by a user and recognized within the data processing system as an indication of the user's ability to read, write, delete, copy or append a selected record. One example of such a system is disclosed in U.S. Pat. No. 4,799,258. Further, several known techniques exist for storing such "keys," "passwords" or other secure data within secure storage devices within a data processing system. For example, U.S. Pat. No. 4,949,927 discloses a method for providing a security module for physically protecting such sensitive data. Similarly, U.S. Pat. No. 4,759,062 discloses a method for protecting sensitive data, such as private security codes.

Each of the methods described above permits the storage and utilization of sensitive or private data; however, none of these publications teaches a technique whereby the functional characteristics of a data processing system may be selectively altered. Systems do exist for enabling or disabling electronic equipment utilizing "keys" or other similar devices. Primarily such systems are directed to enabling or disabling reception of television or CATV signals within a television receiver. For example, see U.S. Pat. Nos. 4,577,224 and 4,471,379.

From U.S. Pat. No. 5,200,999 a public key cryptosystem key management based on control vectors is known. This serves for encrypting the public and private keys of a cryptographic asymmetric key (public key) algorithm, when these keys are stored outside the secure boundary of the cryptographic facility (i. e., cryptographic hardware) and for decrypting these keys when they are processed or used within the secure boundary of the cryptographic facility. The encrypted keys may be kept in a cryptographic key data set belonging to the cryptographic system software or they may be managed by the cryptographic application programs that use the keys. The public and private keys are encrypted by a system master key stored in clear form within the secure boundary of the cryptographic facility.

U.S. Pat. No. 4,908,861 discloses a data authentication method using modification detection codes based on a public one way encryption function. According to this method a message of arbitrary length is transformed into a block of fixed length defined modification detection code (MDC). Although there are a large number of messages which result in the same MDC, because the MDC is a many-to-one function of the input, it is required that it is practically not feasible for an opponent to find them. In analyzing the methods, a distinction is made between two types of attacks, i.e., insiders (who have access to the system) and outsiders (who do not).

From TDB No. 11, April 1992, pages 376–383 the solution of a certain concurrence related to the RSA public key cryptosystem is addressed. The underlaying RSA public key cryptosystem is known from R. L. Rivest, A. Shamir and L. Adleman, "A Method for Obtaining Digital Signatures and Public Key Cryptosystems," Communications of the ACM 21, 2 (February 1978). In TDB Vol. 36, No. 10, October 1993, pages 413–416 a protocol is disclosed for one-way authentication in the asymmetric model. In this protocol, a prover convinces a verifier that the prover knows the factus of large composite number N. The prover does this in a way that it does not reveal the factors of N. This protocol is useful for the software licensing problem.

In electronic design, Apr. 17, 1995, page 96, an overview of different encryption techniques, especially the RSA public key concept is given.

In U.S. Pat. No. 5,365,587 a system is disclosed for selectively altering the functional characteristics of a data processing system without physical or mechanical manipulation by providing an access code from a remote personal identification number generator to a secure controller and store of the computer system. This enables remote authorization of change in function of the computer system, such as performance tune up, speeding clock time, changing function and like changes. The computer system is first manufactured having a predetermined set of functional characteristics. A multibit alterable code which includes a functional characteristic definition is then initially loaded into physically secure, nonvolatile memory within the data processing system, utilizing an existing bus, or a fusible link which may be opened after loading is complete. The functional characteristic definition is loaded from nonvolatile memory into a nonscannable register within a secure portion of a control logic circuit each time power is applied to the data processing system and the definition is then utilized to enable only selected functional characteristics. Alternate functional characteristics may thereafter be selectively enabled by entering a security code which matches one of a number of preloaded codes and an encoded alternate functional characteristic definition. The alternate functional characteristic definition may be enabled on a one-time, metered, or regularly scheduled basis and variable capability data processing systems may be implemented in this manner utilizing a single manufactured system, without the necessity of manufacturing and storing multiple data processing system models.

In summary, the prior art is silent as to the usage of encryption techniques to verify the configuration of a computer system.

The underlying problem of the invention is to provide a method for verifying the configuration of a computer system, a method for encryption of data being expressive of the configuration of a computer system and a computer system for carrying out such a method.

The problem of the invention is solved by the features set forth in the independent claims.

Data which is expressive of the configuration of a computer system advantageously is encrypted during manufacturing of the computer system. This is done by using an identifier which is assigned to the computer system or a component thereof during manufacturing. The private key which is used for the encryption of the encoded data is only known to the manufacturer of the computer system.

The RSA cryptosystem preferably is used for encryption of the encoded data (cf. R. L. Rivest, A. Shamir and L. Adleman "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems", Communications of the ACM, February 1978, Vol. 21, No. 2). For encoding the data by means of the identifier, the identifier can for example simply be added to the data. For decoding the identifier is subtracted later on from the encoded data. Also the DES method can be used whereby the identifier of the computer system is employed as a secret key (cf. Cheryl Ajluni, "Security Techniques Ensure Privacy", Electronic Design, Apr. 17, 1995, page 98).

The encrypted data can be stored in any kind of storage device of the computer system, for example on an EPROM or on a diskette. The encrypted data can already be stored in the computer system during manufacturing. However, it is also possible to transmit the encrypted data to the computer system via a telephone line, ISDN or other telecommunication means when the computer system is already installed at the customer. This is advantageous if the configuration of the computer system is changed, a new set of encrypted data reflecting the changed configuration of the computer system is transmitted and stored on the storage device of the computer system. This saves a service engineer of the manufacturer of the computer system from travelling to the customer site in order to change the encrypted data according to the new configuration.

Once the encrypted data is stored on a storage device of the computer system, the encrypted data is used for verifying the configuration. This serves to protect the computer system against unauthorized changes of its configuration. This can be a requirement for technical reasons or can serve as asset protection for the manufacturer of the computer system.

The first step for verifying the configuration is to receive the encrypted data. This is accomplished by reading the encrypted data from the storage device of the computer system on which the encrypted data has been stored during manufacturing or by receiving the encrypted data via a telecommunications link directly from the manufacturer. Thereafter the encrypted data is decrypted, preferably using a public key of the RSA cryptosystem. This yields the decoded data which has been encoded by means of the identifier. The identifier is available in the computer system, preferably in electronically readable form. For example, the identifier can be stored in one of the components of the computer system by a number of fuses which are blown according to the identifier of that component. This identifier can also serve as an identifier for the entire computer system. In order to prevent the cloning of the computer system with another computer system having another identifier, the identifier has to be unchangeable. This can be guaranteed by storing the identifier in a component of the computer system which is exclusively produced by the manufacturer of the computer system. It is also preferable that the identifier is unique. However, absolute uniqueness is normally not required. If for example every thousandth computer system which is manufactured by the same manufacturer has an identical identifier, this would be a sufficient degree of security.

If the private and the public key match and if the same identifier is used for the encoding and decoding of the data then this yields the data which is expressive of the configuration of the computer system stored during manufacturing. The configuration data of the computer system is also stored on a storage device of the computer system in unencoded form. These configuration data are compared to the decoded data. If there is a match between the decoded data and the unencoded configuration data this means that the customer is authorized to use this configuration of the computer system. Preferably, this method for verifying of the configuration is carried out by means of microcode every time the computer system is booted.

Alternatively—instead of decoding the decrypted data—it is also possible to encode the configuration data which is stored in an encoded form in the computer system and to compare the encoded data with the encoded configuration data.

If the data and the configuration data do not match it is possible not to enable or to disable the entire system. An alternative is to enable the computer system only insofar as the data and the configuration data match. This can be advantageous in the following scenario: A customer is assumed to have purchased a memory expansion card for the computer system from a third party which is not authorized by the manufacturer of the computer system to deliver such memory expansion cards. The memory expansion card is inserted into the computer system and the configuration data which is normally stored on a hard disk of the computer system is changed correspondingly. This change of the configuration data can be carried out easily since the configuration data is present in the computer system in an unencoded and unprotected form. However, when the computer system is booted the encrypted data which is provided by the manufacturer of the computer system is decoded and decrypted and subsequently compared to the unencoded configuration data. In the example considered here the data and the configuration data match with the exception of the memory expansion card. As a consequence this memory expansion card is ignored and the computer system is initialized during the booting procedure so that the memory expansion card is not addressable. This can also be done with other functional characteristics of the computer system, such as the cycle time. A method for selectively altering the functional characteristics as such is known from U.S. Pat. No. 5,365,587.

In order to circumvent the method for verifying of a configuration of a computer system a third party could analyze the microcode which serves to carry out this method and find a way to bypass the corresponding portions of the microcode. This can be prevented if the microcode is also protected against tampering. This can be accomplished for example by special check sums of the microcode which are predefined and checked by a special routine periodically. From IBM Technical Disclosure Bulletin, No. 9, February 1992, pages 188–191 a mechanism for Trusted Computing Base definition and checking is known which could also be used to prevent a third party from altering or bypassing the microcode.

BRIEF DESCRIPTION OF THE DRAWINGS

One way of carrying out the invention is described in detail below with reference to drawings which illustrate only one specific embodiment, in which:

FIG. 1 is a flow chart illustrating the method for encryption of data;

FIG. 2 is a flow chart illustrating the method for verifying of the configuration of the computer system;

FIG. 3 is a flow chart illustrating an alternative method for verifying of the configuration of the computer system; and

FIG. 4 is a schematic block diagram of a computer system which may be utilized to implement the method and system of the present invention.

DETAILED DESCRIPTION OF THE INVENTION

In step 1 of the flow chart shown in FIG. 1 the configuration of the computer system is defined by the manufacturer of the computer system. This results in data which is expressive of the specific configuration of the computer system considered here. In step 2 a unique and unchangeable identifier of the computer system is defined. The identifier is stored in a portion of the computer system where it is protected against tampering. In this example the identifier is stored on the clock chip of the computer system which is a specialized card which can not be produced or delivered by any third party. The clock chip is personalized by blowing a number of fuses according to the identifier. The identifier is used in step 3 to encode the data which is expressive of the configuration of the computer system. This is accomplished—in the prefered embodiment considered here—by by simply adding the identifier which is assumed to be a string of binary data to the data which is expressive of the configuration. Any algorithm using the identifier as a key could be used for encoding of the data, if the identifier can also be used for decoding of the encoded data according to the algorithm.

In step 4 the private key required for the encryption algorithm is fetched. Subsequently, the encoded data is encrypted in step 5 by means of the private key. The resulting encrypted data of step 5 is stored in the computer system in step 6 for example in an EPROM of the computer system.

When the computer system is booted on the customer site this result of the encryption of step 5 is outputted by the EPROM (step 7). Then the public key which is required for decryption of result is fetched from the hard disk of the computer system. In step 9 the decryption of the result is carried out by means of the public key. In step 10 the identifier is read from the clock chip of the computer system. The result obtained by the decryption of step 9 is decoded by means of the identifier in step 11. In the example considered here this is accomplished by subtracting the identifier from the result obtained in step 9.

In step 12 the configuration data is read from the hard disk of the computer system. The configuration data is compared to the result of step 11 in step 13. If there is a perfect match the booting procedure normally continuous and all the hardware is enabled as defined by the configuration data in step 14. If there is no perfect match one possibility is to simply stop the booting procedure so that the entire computer system is not usable (step 15). Another possibility is to selectively enable the functional characteristics of the computer system insofar as there is a match between the result obtained in step 11 and the configuration data read in step 12.

FIG. 3 shows an alternative way of carrying out the method illustrated in FIG. 2. In step 16 of FIG. 3 the result obtained in step 6 is outputted from the EPROM of the computer system. The following steps 17 and 18 of FIG. 3 corresponds to the steps 8 and 9 of FIG. 2 and serve to decrypt the result which was outputted in step 16.

In step 19 the configuration data is read from the hard disk of the computer system. Step 19 in FIG. 3 corresponds to step 12 in FIG. 2. Subsequently the configuration data which is read in FIG. 19 is encoded in step 21 by means of the identifier which is fetched from the clock ship in step 20. The results of the decryption of step 18 and the encoding of step 21 are compared in step 22. Based on the comparison carried out in step 22 it is decided in step 23 whether the decrypted data of step 18 and the encoded data of step 21 match. If there is a match step 24 is carried out which corresponds to step 14 of FIG. 2. If there is no match step 25 is carried out which corresponds to step 15 of FIG. 2.

For an overview of the computer system, the data processing system which can be selectively altered for functional characteristics without physical or mechanical manipulation, refer to FIG. 4. FIG. 4. depicts a high level block diagram of a data processing system which may be utilized to implement the method and system of the present invention. As illustrated, the data processing system includes a computer 26 having a data link 27 and an operator console 28 coupled in a manner well known in the art. Many of the high level components within computer 26 are depicted within FIG. 4 including main store 29, which serves as the main electronic storage within computer 26, and a central electronic complex 30 is also depicted. As will be explained in greater detail herein, central electronic complex 30 may include multiple multi chip modules which serve to perform the various functions of the central electronic complex, or alternately, central electronic complex 30 may be provided with a single high density circuit and include integrated circuit devices equivalent to several million transistors.

A service processor 31 is provided and is preferably coupled between operator console 28 and central electronic complex 30 to provide access to the functions and circuitry therein. A power supply 32 and input/output channels 33 are also typically provided in such a computer system, as those skilled in the art will appreciate.

Still referring to FIG. 4, the high level segments of central electronic complex 30 are illustrated. In a modern mainframe computer such as the International Business Machines Corporation System/390 (System/390 is a registered trademark of International Business Machines Corporation) the central electronic complex typically includes 1 or more multiple chip modules which serve to address various functions within a central electronic complex. As illustrated within FIG. 4, central electronic complex 30 includes an SC module 34 which preferably serves to buffer and control the flow of data between main store 29, input/output module 35 and various processes within computer 26. Input/output module 35 preferably serves to control and buffer data between input/out channels 33 and main store 29 in a manner well known in the art. Similarly, B-module 36 is provided to buffer and control instructions and data for the processor and CP-module 37 serves to execute instructions within computer 26. As those skilled in the art will appreciate, each of these multi chip modules 34, 35, 36 and 37 constitutes a highly complex electronic module which may include more than 100 integrated circuit devices, each equivalent to thousands or millions of transistors.

Additionally, as those skilled in the art will appreciate, a translation look aside buffer (TLB) is also provided and is utilized, in a manner well known in the art, to translate virtual memory address into real memory addresses within main store or other locations within computer 26.

Mainframe computers such as the International Business Machines Corporation System/390 may include multiple

levels of functional capability which may be provided by altering the range of memory that may be accessed within a particular computer system, the number or percentage of processors which are enabled within a particular computer, the amount of usable cache memory within a particular computer and the processors speed and/or capability provided within a particular computer. Thus, by providing computer **26** with the capability of all these functional characteristics during the manufacturing process a selected subset or variations of those functional characteristics may be enabled utilizing the method and system of U.S. Pat. No. 5,365,587.

Furthermore, the computer **26** comprises a clock chip **39** having the identifier stored therein by means of a number of fuses. The identifier is unique, unchangeable and electronically readable by the service processor **31** to which the clock chip **39** is connected. The computer system shown in FIG. **4** further comprises a hard disk **38**. The hard disk **38** serves to receive the encrypted data which is encrypted according to the method of the invention. The encrypted data can be stored on the disk **38** during manufacturing. If the configuration of the computer system is changed a new set of encrypted data can be stored on the disk **38** at the customer site. It is also possible to input the updated encrypted data via the channels **33** into the computer **26** and to store the updated encrypted data on the disk **38** under the control of the service processor **31**. Every time the computer system depicted in FIG. **4** is booted the service processor **31** accesses the disk **38** in order to read the encrypted data. The service processor **31** also accesses the disk **38** to read the public key needed for decryption according to the RSA method. The decryption of the encrypted data read from the disk **38** is carried out by the service processor **31** whereby usage is made of the public key. When the service processor **31** accesses the clock chip **39** in order to read the identifier.

Subsequently, the service processor **31** decodes the decrypted data and compares the result with the actual configuration data stored in unencrypted form on the disk **38**. Alternatively the service processor **31** is programmed to encode the actual unencrypted configuration data by means of the identifier in order to compare the decrypted data and the encoded actual configuration data.

In response to the comparison carried out in service processor **31** the computer system is enabled according to the actual configuration data if a perfect match occurs. The selective enabling of functional characteristics of the computer system **26** advantageously is carried out according to the method disclosed in U.S. Pat. No. 5,365,587. If the comparison carried out by the service processor **31** reveals that there is no perfect match this can cause the interruption of the booting procedure so that the entire computer system is disabled. Alternatively only those functional characteristics of the actual configuration data stored on disk **38** which match the encrypted data are enabled selectively.

It is also possible to program the service processor **31** to carry out the method of the invention not during the initialization of the computer system **26** but during normal operation. If a mismatch occurs this can lead to a shut down of the entire computer system **26**. Alternatively only those features of the computer system which correspond to configuration data which do not match the decrypted configuration data are disabled.

Though preferred embodiments have been depicted and described in detail herein, it will be apparent to those skilled in the relevant art, both now and in the future, that various modifications, additions, improvements and enhancements may be made without departing from the spirit of the invention, and these are therefore considered to be within the scope of the invention as defined in the following claims.

I claim:

1. A method for the verifying of data, said data being expressive of a configuration of a computer system, said computer system or a component thereof having an identifier, said method comprising the steps of:

a) encoding said data by an encoding method, said encoding method using said identifier to encode said data to produce encoded data;

b) encrypting said encoded data by an encryption method, said encryption method using a private key;

c) receiving said encrypted data;

d) decrypting said encrypted data using a public key, said decrypting of said encrypted data producing said encoded data, said encoded data capable of being decoded via a decoding method using said identifier;

e) decoding said encoded data via said decoding method using said identifier to decode said encoded data producing said decoded configuration data; and

f) comparing said decoded configuration data with said configuration data stored in said computer system, said configuration data being expressive of the configuration of the computer system.

2. A method according to claim **1** wherein if said comparing of said decoded data with said configuration data stored in said computer system indicates a match therebetween, the method further includes the step of:

enabling a functional characteristic of the computer system.

3. A method according to claim **2** wherein said functional characteristic is a cycle time for the computer system.

4. A method according to claim **2** further including the steps of:

receiving said encrypted data;

decrypting said encrypted data using a public key, said decrypting of said encrypted data producing said encoded data;

decoding said encoded data via a decoding method using said identifier; and

comparing said decoded data with a copy of said data expressive of the configuration of the computer system, said copy being stored in said computer system.

5. A method according to claim **1** further including the steps of:

receiving said encrypted data;

decrypting said encrypted data using a public key, said decrypting of said encrypted data producing said encoded data;

encoding, using an encoding method, a copy of said data expressive of the configuration of the computer system, said copy being stored in said computer system, said encoding method using said identifier to produce encoded configuration data; and

comparing said encoded data with said encoded configuration data.

6. A method according to claim **1** for selectively enabling the functional characteristics of said computer system, said method comprising the steps of:

enabling one or more of said functional characteristics of said computer system if said comparing step indicates a match as regards said functional characteristics.

7. A method for the verifying of data, said data being expressive of a configuration of a computer system, said

computer system or a component thereof having an identifier, said method comprising the steps of:

a) encoding said data by an encoding method, said encoding method using said identifier to encode said data to produce encoded data;

b) encrypting said encoded data by an encryption method, said encryption method using a private key;

c) receiving said encrypted data;

d) decrypting said encrypted data using a public key, said decrypting of said encrypted data producing said encoded data;

e) encoding said configuration data stored in said computer system by said encoding method, said encoding method using said identifier to encode said stored configuration data so as to produce encoded configuration data, said stored configuration data being expressive of the configuration of the computer system; and

f) comparing said encoded data with said encoded configuration data.

8. A method according to claim **7** wherein if said comparing of said encoded data with said encoded configuration data indicates a match therebetween, the method further includes the step of:

enabling a functional characteristic of the computer system.

9. A method according to claim **8** wherein said functional characteristic is a cycle time for the computer system.

10. A method according to claim **7** wherein if said comparing of said decoded configuration data with said stored configuration data indicates a match therebetween as regards a functional characteristic of the computer system, the method further includes the step of:

enabling said functional characteristic of the computer system.

11. A computer system comprising:

means for the reception of encrypted data, said encrypted data being encrypted via an encryption method using a private key;

means for decryption of said encrypted data to yield encoded data, said means for decryption using a public key;

means for decoding said encoded data to yield decoded data, said means for decoding comprising means for accessing an identifier, said identifier being associated with said computer system or with a component thereof; and

means for comparing said decoded data with configuration data stored in said computer system, said stored configuration data being expressive of the configuration of the computer system.

12. A computer system according to claim **11** further comprising:

enabling means for enabling a functional characteristic of the computer system;

wherein said enabling means is responsive to said means for comparing said decoded data with said configuration data stored in said computer system to enable said functional characteristic if there is a match therebetween.

13. A computer system according to claim **12** wherein said functional characteristic is a cycle time for said computer system.

14. A computer system according to claim **11** further comprising means for selectively enabling a functional characteristic of said computer system.

15. A computer system comprising:

means for the reception of encrypted data, said encrypted data being encrypted via an encryption method using a private key;

means for decryption of said encrypted data to yield encoded data, said means for decryption using a public key;

means for encoding configuration data stored in said computer system, said stored configuration data being expressive of the configuration of the computer system, said means for encoding using an identifier to encode said configuration data, said identifier being associated with said computer system or with a component thereof; and

means for comparing said encoded data with said encoded configuration data.

16. A computer system according to claim **15** further comprising:

enabling means for enabling a functional characteristic of the computer system;

wherein said enabling means is responsive to said means for comparing said encoded data with said encoded configuration data to enable said functional characteristic if there is a match therebetween.

17. A computer system according to claim **16** wherein said functional characteristic is a cycle time for said computer system.

18. A computer system according to claim **15** further comprising means for selectively enabling a functional characteristic of said computer system.

19. An apparatus for the encryption of data, said data being expressive of a configuration of a computer system, said computer system or a component thereof having an identifier, said apparatus comprising:

means for encoding said data by an encoding method, said encoding method using said identifier to produce encoded data; and

means for encrypting said encoded data by an encryption method, said encryption method using a private key.

20. An apparatus according to claim **19** wherein said encryption method includes applying the RSA algorithm.

* * * * *

UNITED STATES PATENT AND TRADEMARK OFFICE

CERTIFICATE OF CORRECTION

PATENT NO. : 5,982,899
DATED : November 9, 199
INVENTOR(S) : Probst

It is certified that error appears in the above-identified patent and that said Letters Patent is hereby corrected as shown below:

Title page, item [54] and col. 1, line 2, delete "THE" and insert --OF A--.

Signed and Sealed this

Twenty-fourth Day of October, 2000

Attest:

Q. TODD DICKINSON

Attesting Officer

Director of Patents and Trademarks

US006212639B1

(12) **United States Patent**
Erickson et al.

(10) **Patent No.: US 6,212,639 B1**
(45) **Date of Patent: Apr. 3, 2001**

(54) **ENCRYPTION OF CONFIGURATION STREAM**

(75) Inventors: **Charles R. Erickson**, Fremont; **Danesh Tavana**, Mountain View; **Victor A. Holen**, Los Gatos, all of CA (US)

(73) Assignee: **Xilinx, Inc.**, San Jose, CA (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

(21) Appl. No.: **09/342,336**

(22) Filed: **Jun. 29, 1999**

Related U.S. Application Data

(62) Division of application No. 08/703,117, filed on Aug. 26, 1996.

(51) **Int. Cl.**[7] **H04K 1/00**

(52) **U.S. Cl.** **713/200**; 713/166; 713/187; 713/189; 713/196; 713/193; 380/268; 326/38

(58) **Field of Search** 713/200, 166, 713/187, 189, 190, 193; 380/268; 326/38

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,081,675		1/1992	Kittirutsunetorn	380/4
5,349,249	*	9/1994	Chiang et al.	307/465
5,388,157	*	2/1995	Austin	380/4
5,406,627		4/1995	Thompson et al.	380/20
5,748,734	*	5/1998	Mizikovsky	380/21
5,768,372	*	6/1998	Sung et al.	380/3
5,915,017	*	6/1999	Sung et al.	380/3

OTHER PUBLICATIONS

"The Programmable Logic Data Book," pp. 2–25 through 2–46, Xilinx, 1994, San Jose, California.
Datasheet for XC5200 FPGA from Xilinx.
Schneier, Applied Cryptography, 2nd edition, pp. 4, 5 and 32–36, Oct. 1995.*

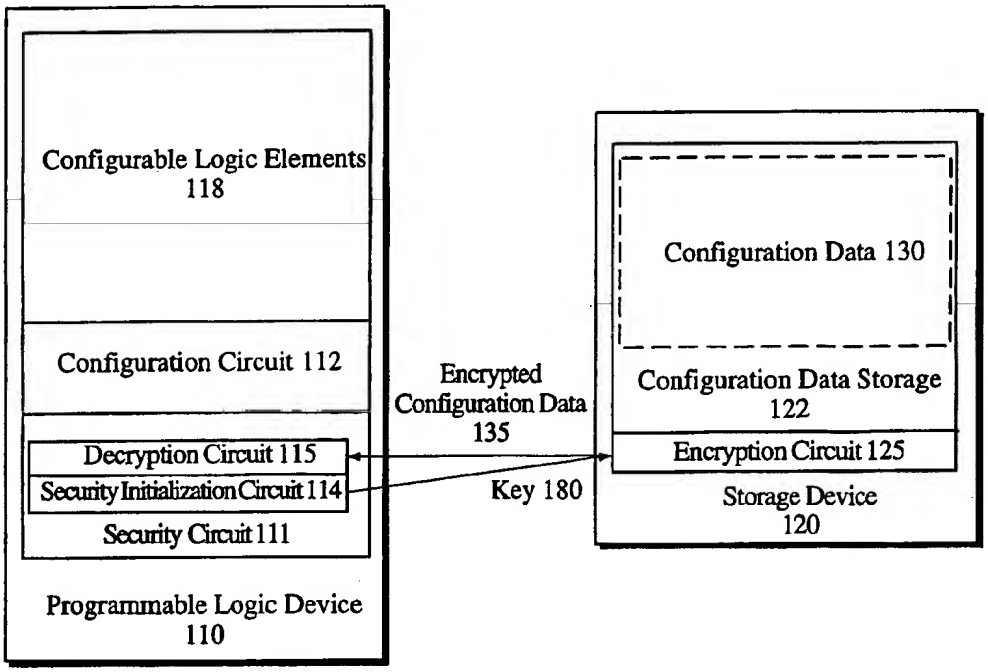
* cited by examiner

Primary Examiner—Tod R. Swann
Assistant Examiner—Paul E. Callahan
(74) *Attorney, Agent, or Firm*—William L. Paradice, III; Edel M. Young

(57) **ABSTRACT**

A method of communicating encrypted configuration data between a programmable logic device (PLD) and a storage device is included in one part of the invention. The method includes the following steps. Transmit encrypted configuration data stored in a storage device to the PLD. Decrypt the encrypted configuration data to generate a copy of the configuration data in the PLD. Configure the PLD using the copy of the configuration data. In one embodiment, the PLD transmits a key to the storage device. In another embodiment the key is separately entered into the storage device and the PLD and never transmitted between the PLD and the storage device. In another embodiment, the key is entered only into the PLD. The key is used to encrypt the configuration data.

2 Claims, 4 Drawing Sheets



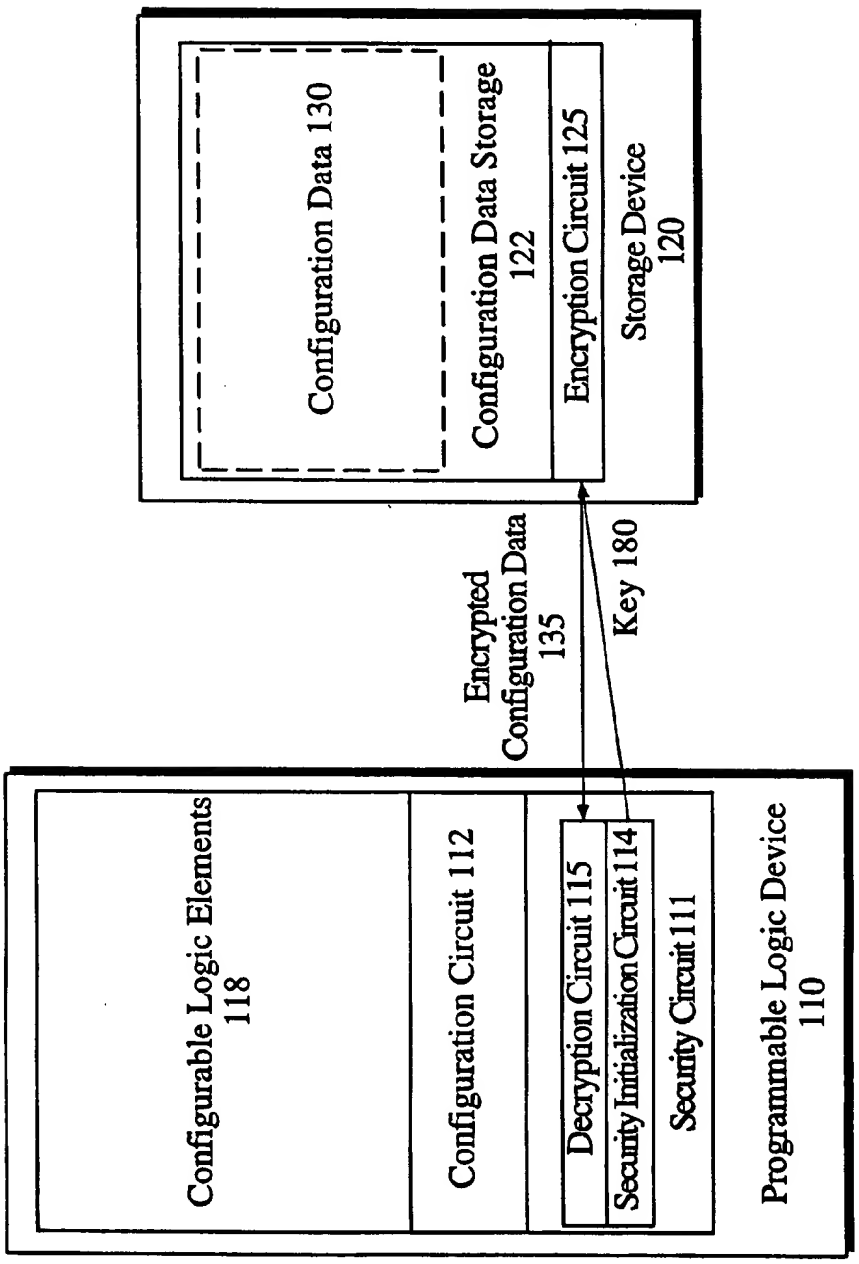

Configurable Logic Elements
118
Configuration Circuit 112
Decryption Circuit 115
Security Initialization Circuit 114
Security Circuit 111
Programmable Logic Device
110
Encrypted
Configuration Data
135
Key 180
Configuration Data 130
Configuration Data Storage
122
Encryption Circuit 125
Storage Device
120

FIGURE 1

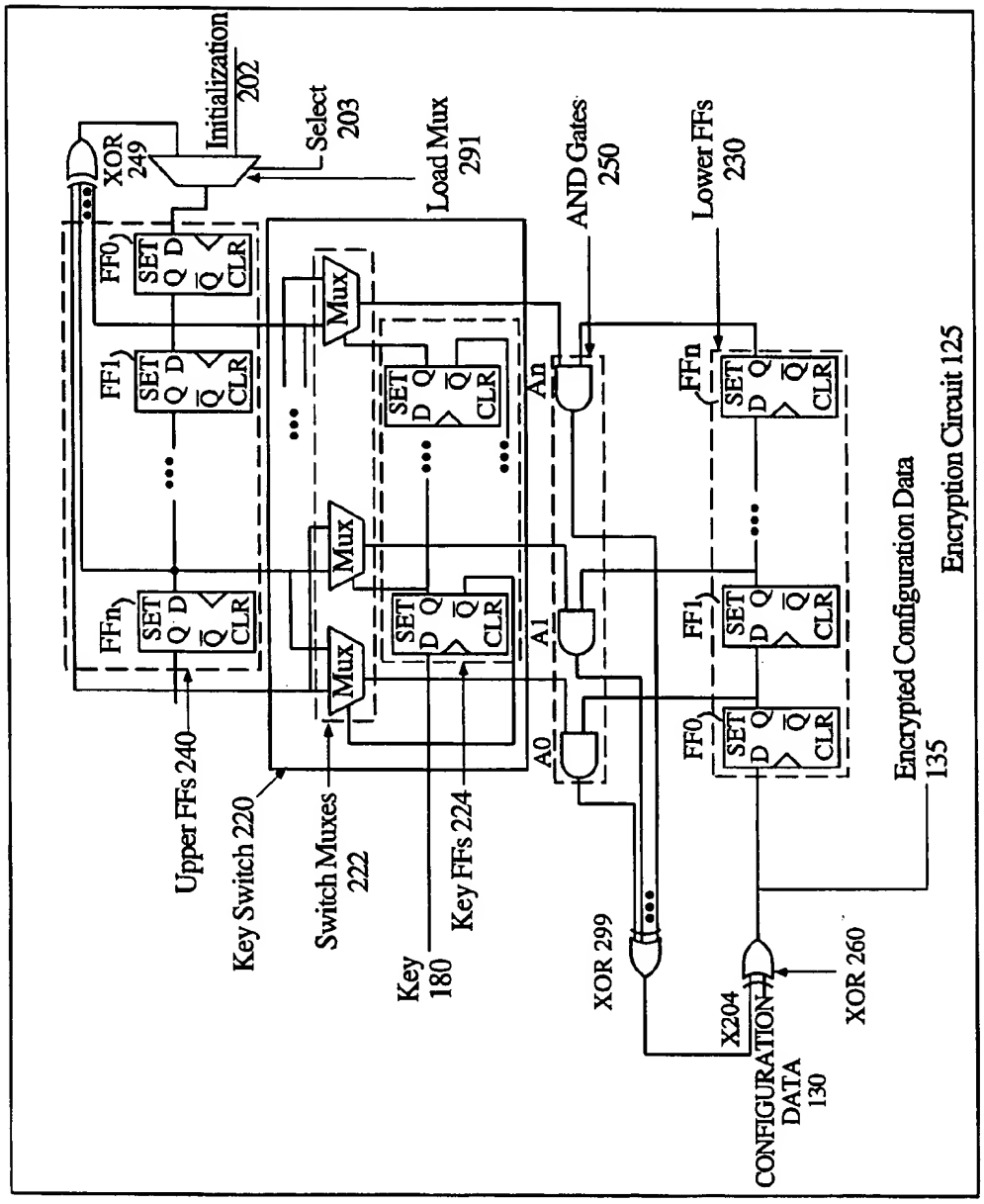

Initialization 202
Select 203
Load Mux 291
AND Gates 250
Lower FFs 230
XOR 249
FF0
FF1
FFn
SET
CLR
Mux
An
A1
A0
Upper FFs 240
Key Switch 220
Switch Muxes 222
Key FFs 224
Key 180
XOR 299
X204
CONFIGURATION DATA 130
XOR 260
Encrypted Configuration Data 135
Encryption Circuit 125

FIGURE 2

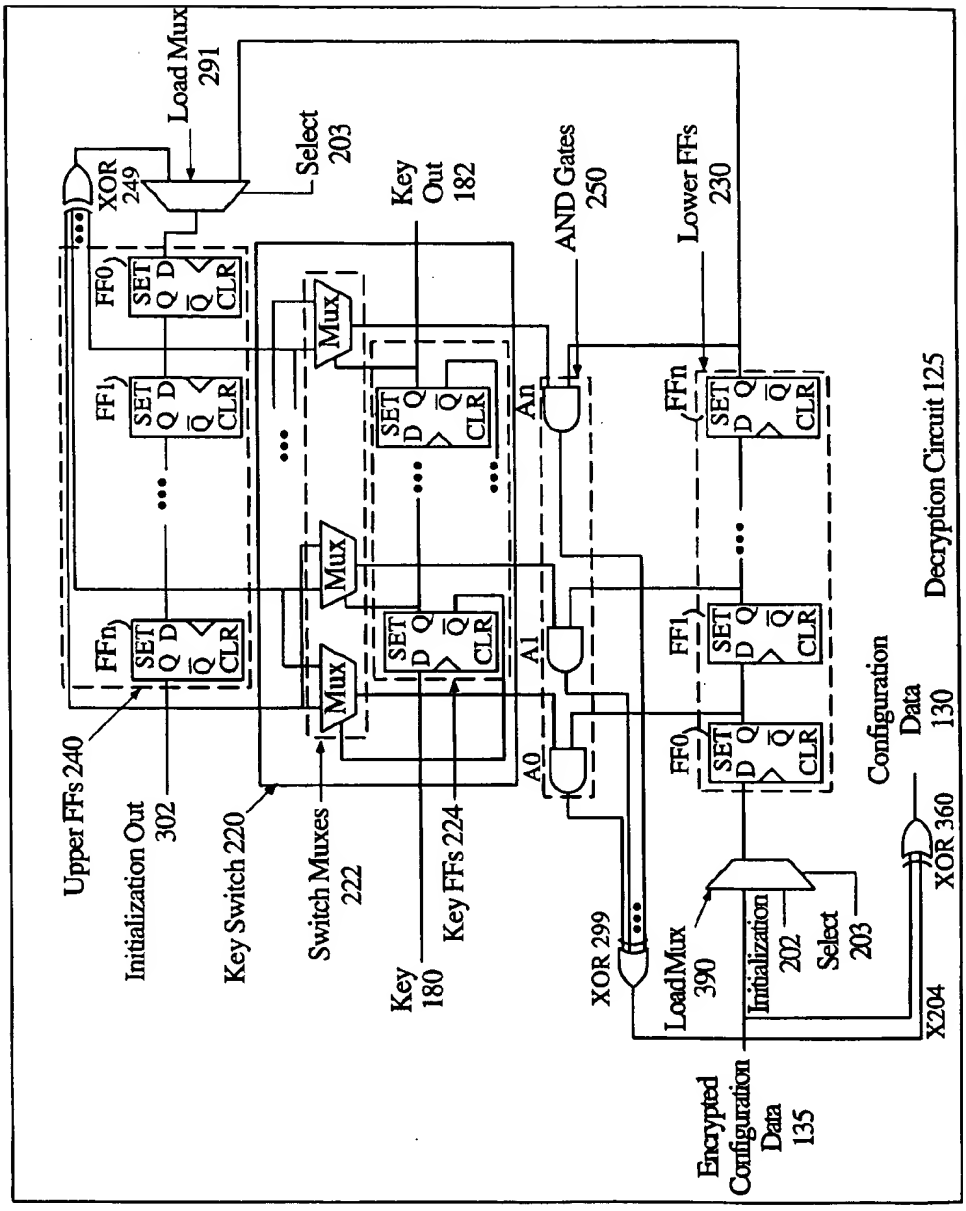

Load Mux 291
Select 203
XOR 249
Key Out 182
AND Gates 250
Lower FFs 230
FF0
FF1
FFn
SET
CLR
Mux
An
A1
A0
Upper FFs 240
Initialization Out 302
Key Switch 220
Switch Muxes 222
Key 180
Key FFs 224
XOR 299
LoadMux 390
Initialization 202
Select 203
Encrypted Configuration Data 135
Configuration Data 130
XOR 360
X204
Decryption Circuit 125

FIGURE 3

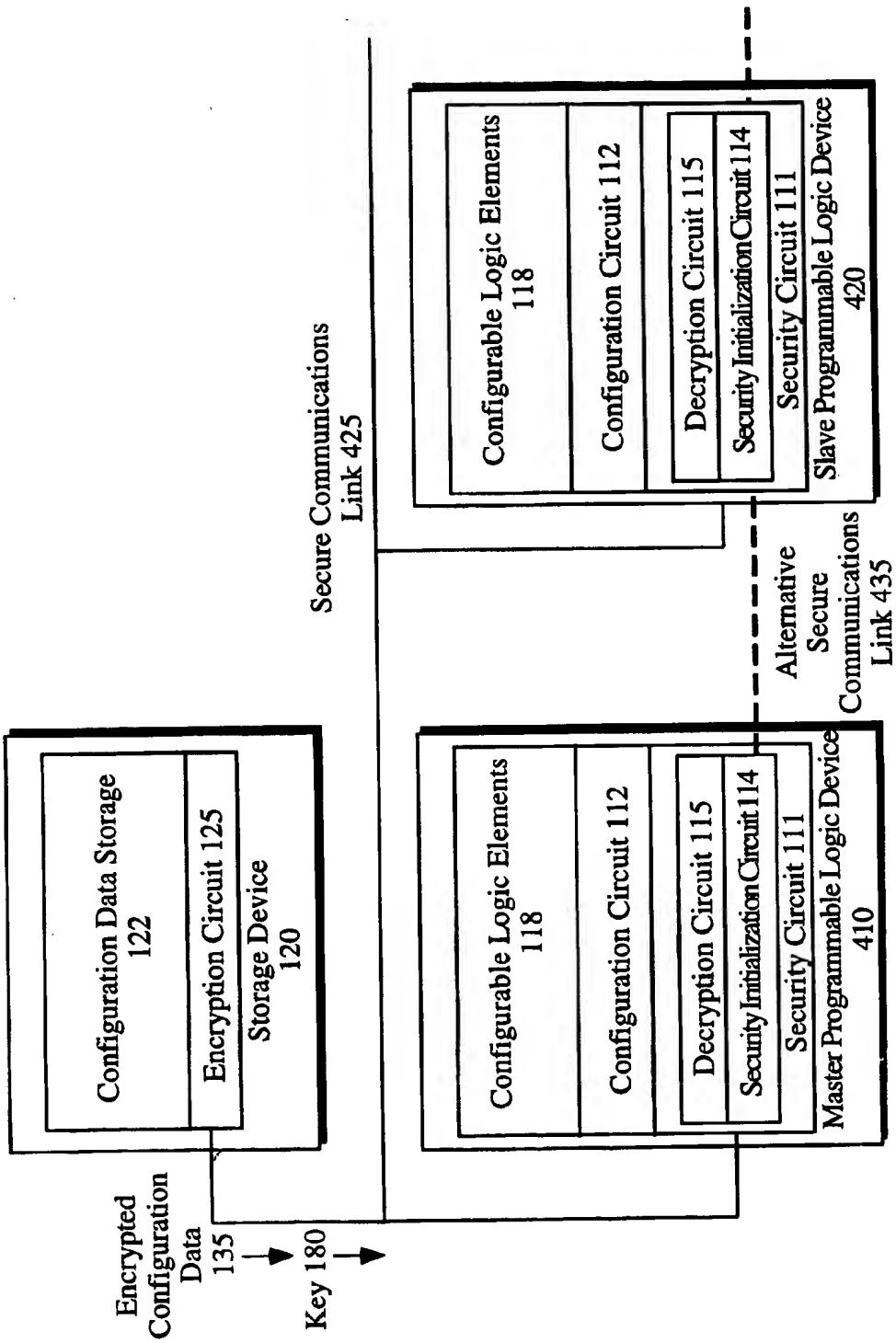

Configuration Data Storage 122
Encryption Circuit 125
Storage Device 120
Encrypted Configuration Data 135
Key 180
Secure Communications Link 425
Configurable Logic Elements 118
Configuration Circuit 112
Decryption Circuit 115
Security Initialization Circuit 114
Security Circuit 111
Master Programmable Logic Device 410
Alternative Secure Communications Link 435
Slave Programmable Logic Device 420

FIGURE 4

ENCRYPTION OF CONFIGURATION STREAM

RELATED APPLICATIONS

This application is a divisional of U.S. patent application Ser. No. 08/703,117 entitled "Configuration Stream Encryption" filed Aug. 26, 1996, which is incorporated herein by reference.

BACKGROUND OF THE INVENTION

1. Field of the Invention

This invention relates to the field of circuit design. In particular, the invention relates to a method and apparatus for securing data used to configure a programmable logic device.

2. Background Information

Programmable Logic Devices (PLDs) are a class of devices that allow a user to program a device to perform the function of a particular circuit. Examples of PLDs are FPGAs (Field Programmable Gate Arrays) and EPLDs (Erasable Programmable Logic Devices).

To use a PLD, a user captures a circuit design using any of several design capture tools. The user then uses software tools to convert the captured design into a device specific bitwise representation. The bitwise representation is stored in a storage device, such as an EPROM. Upon startup, the storage device supplies the bitwise representation to the PLD, thereby enabling the PLD to perform the function of the circuit design. The PLD, having read in the bitwise representation, then performs the function of the circuit design.

By the time the bitwise representation is created, significant amounts of time and effort have been expended. To encourage individuals and companies to continue to invest in the research and development of new circuit designs, it is desirable to provide some method of protecting the circuit designs from illegal copying and use.

To make an illegal copy of the circuit design, as implemented in the programmable logic device, one need only make a copy of the bitwise representation stored in the storage device. The copied bitwise representation can then be illegally used with other programmable logic devices. Therefore, it is desirable to make it more difficult to copy the bitwise representation of the circuit design.

Additionally, some types of PLDs support multiple configuration modes. For example, the XC4000™ series FPGAs, available from Xilinx, Inc. of San Jose, Calif., supports multiple configuration modes. The 1994 Xilinx Data Book, page 2–25 through page 2–46, describes the unsecured configuration modes for the XC4000™ FPGA product family. Therefore, it is desirable to have secure configuration of PLDs that have multiple configuration modes. Of course no system can be absolutely secure from all potential unauthorized access, therefore, the term "secure" is used to mean more secure than systems without any security.

Some PLDs can be chained together for the purpose of configuration. After one PLD is configured, the configuration data is passed to the next PLD in the chain. Therefore, it is desirable to support the secured configuration of multiple chained PLDs.

SUMMARY OF THE INVENTION

A method and apparatus for encrypting the information used in configuring a programmable logic device is described.

A method of communicating encrypted configuration data between a programmable logic device (PLD) and a storage device is included in one part of the invention. The method includes the following steps. Transmit encrypted configuration data stored in a storage device to the PLD. Decrypt the encrypted configuration data to generate a copy of the configuration data in the PLD. Configure the PLD using the copy of the configuration data. In one embodiment, the PLD transmits a key to the storage device. In another embodiment, the manufacturer, user, or someone else, stores a key in the storage device and in the PLD. In both embodiments, the key is used to encrypt the configuration data.

In one embodiment, the storage device includes an encryption circuit. The encryption circuit generates a bit of the encrypted configuration data, D*, from a bit of the configuration data, D, using the relationship: $D \oplus X = D^*$, where $\oplus$ indicates an exclusive OR logical operation. X is a signal generated from previous bits of the encrypted configuration data. The PLD includes a decryption circuit. The decryption circuit generates a copy of the bit of the configuration data, D, from a bit of the encrypted configuration data, D*, using the relationship: $D^* \oplus X = D$.

In one embodiment, the storage device includes no encryption circuit. The PLD and storage device are used in pairs. A software system (work station) or user generates or supplies a key and sends the key or a related key to the PLD. It generates encrypted configuration data using the key and sends the encrypted configuration data to the storage device. The PLD includes a decryption circuit. The key in the PLD is used by this decryption circuit to decrypt the encrypted configuration data received from the storage device.

In one embodiment, multiple PLDs are chained together during the configuration mode. The storage device transmits the encrypted configuration data to the first PLD in the chain, then to the next PLD.

In one embodiment, each PLD listens to all of the encrypted configuration data until the storage device begins transmitting the encrypted configuration data for that PLD. In another embodiment, the first PLD decrypts the configuration data for itself. When fully programmed, the first PLD passes the encrypted configuration data onto the next PLD in the chain. In this embodiment, the programmed PLD also transfers the current state of its decryption circuit to the next PLD in the chain.

Although many details have been included in the description and the figures, the invention is defined by the scope of the claims. Only limitations found in those claims apply to the invention.

BRIEF DESCRIPTION OF THE DRAWINGS

The figures illustrate the invention by way of example, and not limitation. Like references indicate similar elements.

FIG. **1** illustrates a programmable logic device and storage device having security circuits.

FIG. **2** illustrates an encryption circuit used in a storage device.

FIG. **3** illustrates a decryption circuit used in a programmable logic device.

FIG. **4** illustrates a number of daisy-chained programmable logic devices having security circuits.

DETAILED DESCRIPTION OF THE DRAWINGS

Secure Programmable Logic Device System

FIG. **1** illustrates a programmable logic device (PLD) and storage device having security circuits. In one embodiment

of the invention, the PLD provides the storage device with a key. The storage device then encrypts the bitwise configuration data before transmitting the configuration data to the PLD. The PLD then decrypts the configuration data prior to using the configuration data.

The following paragraph identifies the elements of FIG. **1** and how the elements are connected. FIG. **1** includes a PLD **110** and a storage device **120**. The PLD **110** includes the following elements: a security circuit **111**; a configuration circuit **112**; and a number of configurable logic elements **118**. The security circuit **111** includes a security initialization circuit **114** and a decryption circuit **115**. The storage device **120** includes an encryption circuit **125**. The security circuit **111** connects to the input of the encryption circuit **125**. The encryption circuit **125** connects to the decryption circuit **115**. The storage device **120** also includes a configuration data storage unit **122**. The configuration data storage unit **122** stores the configuration data **130**. The configuration data **130** includes the bitwise representation of the circuit design, as that circuit design is to be implemented by the PLD **110**. The configuration data **130** is what is protected by one embodiment of the invention.

In one embodiment of the invention, the configurable logic elements **118** are programmed as follows. First, the PLD **110** waits until the power supply becomes stable at a predetermined voltage (e.g., at 3.5 volts). Next, a power-on reset step resets some devices in the PLD **110**. Next, the configurable logic elements **118** are reset. Then, the security initialization circuit **114** generates a pseudo-random digital key **180**. In one embodiment, the key **180** is a string of 0's and 1's eight bits long, not all 1's or all 0's. The key **180** is communicated to the encryption circuit **125**. The encryption circuit **125** then uses the key **180** to generate the encrypted configuration data **135** from the configuration data **130**. The storage device **120** transmits the encrypted configuration data **135** to the decryption circuit **115**. The decryption circuit **115** uses the key **180** from the security initialization circuit **114** to decrypt the encrypted configuration data **135** to generate the configuration data **130**. The configuration data **130** is then fed to the configuration circuit **112**. The configuration circuit **112** uses the configuration data **130** to program the configurable logic elements **118**. Importantly, because the PLD **110** generates the pseudo-random key **180** each time it is programmed, and the key **180** is used to encrypt the configuration data **130**, it is ineffective for a person to copy the encrypted configuration data **135** because the encrypted configuration data **135** will be different each time the PLD is configured. To copy the configuration data **130**, a person must copy the encrypted configuration data **135**, must know the key **180**, and must know the technique used to encrypt the encrypted configuration data **135**.

The following paragraphs describe the elements of FIG. **1** in greater detail.

The configurable logic elements **118** perform the functions of the circuit design. In one embodiment of the invention, the configurable logic elements **118** include configurable logic blocks and configurable input/output blocks similar to those in the XC4000™ series FPGAs. The configuration data **130**, in one embodiment, includes a bitwise representation of the circuit design as implemented in a specific XC4000 series FPGA. In one embodiment, the XACT Step™ software tools generate the bitwise representation. Other embodiments of the invention include other PLDs (e.g., XC5200™ FPGA, also available from Xilinx, Inc., FLEX8000™ available from Altera, Inc., of San Jose, Calif.) and use other tools to generate the configuration data **130** (e.g., Max+Plus II™).

The configuration circuit **112** controls the storage of the configurable logic elements **118** and the operation of the security circuit **111**. The configuration circuit **112** also enables the storage of daisy chained PLDs **110**. The decryption circuit **115** decrypts the encrypted configuration data **135** using the key **180** and the initialization data **202** and supplies the decrypted configuration data **130** to the configuration circuit **112**. Analogously, the encryption circuit **125** encrypts data received from the configuration data storage unit **122** to generate the encrypted configuration data **135**. The decryption circuit **115** and the encryption circuit **125** are described in greater detail below.

The security circuit 111 generates keys **180** for use in the encryption process. The use of the keys **180** provide improved security over one embodiment of the invention. In this alternate embodiment of the invention, the configuration data **130** is encrypted by the software used to generate the configuration data **130**, e.g., the configuration data **130** is encrypted by an extension to the XACT Step tools. The encrypted configuration data **135** is then stored in the storage device **120**. In this embodiment, the storage device does not include the encryption circuit **125**, and the security circuit **111** does not generate keys **180** for the storage device **120**. In this embodiment, the PLD **110** simply decrypts the encrypted configuration data **135** generated by the software tools. However, to copy the circuit design as implemented in the PLD **110**, one need only copy the encrypted configuration data **135** and store this data in a storage device **120**. Thus, in one embodiment, at least one pseudo-random key **180** is generated in the PLD **110**. The key **180** is then used by the storage device **120** to encrypt the configuration data **130**; thus, making copying of the configuration data **130** more difficult. In another embodiment, at least a portion of the configuration data **130** is encrypted by the software tools before being stored in the storage device **120** and the encryption circuit **125** further encrypts the already encrypted data. The security circuit **111** performs a complementary double decryption to generate the configuration data needed to program the configurable logic elements **118**.

The storage device **120** is loaded with the configuration data **130**. In one embodiment, the storage device **120** includes an EPROM with the additional encryption circuit **125**. Importantly, in one embodiment, the encryption techniques used in the encryption circuit **125** are difficult to determine. To determine what techniques are being used, one would need to reverse engineer the storage device **120**; a time consuming and difficult task. Other embodiments of the invention include other storage devices such as an EEPROM or a ROM. In one embodiment of the invention, the storage device **120** is replaced by a microprocessor that accesses the configuration data from a storage device (e.g., RAM, ROM) and encrypts the configuration data.

In another embodiment of the invention, the PLD **110** does not include the security circuit **114**. In this embodiment, the PLD **110** and the storage device **120** include storage areas for the key **180**. The manufacturer of the PLD **110** and/or the storage device **120**, the purchaser of the PLD **110** and/or the storage device **120**, or some other person, stores the same key **180** in the PLD **110** and the storage device **120**. In different embodiments, the storage areas include EEPROM memory, mask programmed circuits, anti-fuse circuits, and/or other storage devices for storing the key **180**. Importantly, during configuration, the key **180** is not communicated between the encryption circuit **125** and the decryption circuit **115**. Because the key **180** is never communicated between these two circuits, it is difficult for someone to determine what the key **180** is and

therefore what configuration data is present. In an alternate embodiment of this invention, the security circuit **114** only generates the initialization data **202** (described below). The security circuit **114** transmits the initialization data **202** to the encryption circuit **125** and the decryption circuit **115** instead of the key **180**.

In one embodiment, the security initialization circuit **114** generates multiple keys during the programming of the configurable logic elements **118**. At intervals, the security initialization circuit **114** generates a new key **180**. The new key **180** is then transmitted to the storage device **120**. The new key **180** is then used to encrypt any configuration data **130** transmitted by the encryption circuit **125** until another new key **180** is received or until all of the stored configuration data **130** has been transmitted to the PLD **110** as the encrypted configuration data **135**. In another embodiment, where the security circuit **114** does not transmit a key **180** to the storage device **120**, the security circuit **114** periodically generates new initialization data for the encryption circuit and the decryption circuit. The new initialization data is then used to encrypt and decrypt the configuration data. In one embodiment, the security circuit **111** and the encryption circuit **125** implement a public key cryptography system. In one embodiment, security circuit **111** transmits a PLD public key to the encryption circuit **125**. The encryption circuit **125** then uses the PLD public key to encrypt the configuration data **130**. The security circuit **111** then uses the PLD private key to decrypt the encrypted configuration data **135**. One embodiment of the invention uses the RSA public key cryptography system. Other embodiments of the invention use other public key cryptography systems. In another embodiment, the encryption circuit **125** and the security circuit **111** first secure the communications channel between the two devices using the public key cryptography system and then the storage device **120** uses a symmetric key cryptography system (e.g., DES, available from IBM, Inc.) to transmit the configuration data **130**.

Public key cryptography systems can require a significant amount of circuitry to implement. One embodiment of the invention reduces the amount of circuitry needed in the PLD **110** by programming the PLD **110** to first operate as a public key cryptography circuit to secure the communications link that allows for the use of symmetric key cryptography. The PLD **110** is then reprogrammed using the secure communications link. In this embodiment, the storage device **120** includes two sets of configuration data. The first set of configuration data is not encrypted by the encryption circuit **125**. The first set of configuration data programs the configurable logic elements **118** to operate as a public key cryptography circuit to establish a secure communications link between the PLD **110** and the encryption circuit **125**. The programmed PLD **110** exchanges a secret key with the storage device **120**. The PLD **110** is then reprogrammed with an encrypted version of the second set of configuration data; the encrypted version of the second set of configuration data being generated from the secret key.

Another embodiment of the invention uses a simpler cryptographic system that requires fewer gates to implement and provides adequate security. This embodiment is described in greater detail below.

An Encryption Circuit

FIG. **2** illustrates an encryption circuit used in one embodiment of the invention. The encryption circuit **125** of FIG. **2** uses a relatively small number of gates and provides adequate protection. In the embodiment of FIG. **2**, the relationship between a bit of the configuration data **130**, D, and a bit of the encrypted configuration data **135**. D*, is:

$$D \oplus X = D^* \qquad (EQ.\ 1)$$

where $\oplus$ indicates an exclusive OR operation, X is a signal generated from one or more previous bits of the encrypted configuration data **135**, D*old, and the key **180**. Therefore, to decrypt D*, one need only perform the following operation:

$$D^* \oplus X = D, \qquad (EQ.\ 2)$$

where X remains the same as in equation one.

The following paragraphs describe the elements in FIG. **2** and how they are connected. FIG. **2** includes an encryption circuit **125** having: upper flip-flops **240**, a key switch **220**, AND gates **250**, lower flip-flops **230**, XOR gate **260**, XOR gate **299**, XOR gate **249**, and a load multiplexer **291**. The configuration data **130** and the XOR'd outputs of the AND gates **250** (signal X**204** from XOR gate **299**) are connected to an input of the XOR gate **260**. The output of the XOR **260** is the encrypted configuration data **135**. The encrypted configuration data **135** is fed to the input of the lower flip-flops **230**.

The lower flip-flops **230** include a number of D flip-flops. The first flip-flop has an input connected to receive the encrypted configuration data **135**. The output of the first flip-flop is connected to the input of the second flip-flop. The second flip-flop's output is connected to third flip-flop, etc. Thus, the lower flip-flops **230** form a shift register. In one embodiment, the lower flip-flops **230** include eight D flip-flops. Other embodiments of the invention implement the shift register using different devices (e.g., T flip-flops). Each output of the lower flip-flops **230** is also connected to an input of a different AND gate of the AND gates **250**.

The upper flip-flops **240** form a second shift register, similar to the shift register formed by the lower flip-flops **230**. The outputs of some of the upper flip-flops **240** are fed back, through the XOR gate **249**, into the an input of the load mux **291**. The other input of the load mux **291** is connected to an initialization signal **202**. A select signal **203** connects to the load mux **291** select input. A select signal **203** determines whether the load mux **291** causes a loading of the upper flip-flops **240**, or a feeding back of the XOR'd outputs of the upper flip-flops **240**. How many, and which outputs used as inputs in the XOR gate **249** help scramble the values generated by the upper flip-flops **240**.

The key switch **220** also receives the output of the upper flip-flops **240** and provides additional inputs to the AND gates **250**. The output of each upper flip-flop **240** is connected to two different switch muxes **222**. The select lines of the switch muxes **222** are connected to an output of the key flip-flops **224**. The key flip-flops **224** form a shift register for storing the key **180**. Each output of each of the key flip-flops **224** is connected to the select inputs of two different switch muxes **222**. Each output of each switch mux **222** is connected to an input of an AND gate **250**. The patterns of the connections between the upper flip-flops **240**, the key flip-flops **224**, and the switch muxes **222** help encrypt the configuration data **130**. The outputs of the AND gates **250** are XOR'd together (using XOR gate **299**) to generate the signal X**204**. X**204** is then XOR'd with the configuration data **130**.

The following paragraphs describe the operation of the encryption circuit **125**. Importantly, the encryption circuit **125** supports both an initialization procedure and an encryption procedure.

The initialization procedure prepares the encryption circuit **125** for encrypting the configuration data **130**. That is, prior to beginning to encrypt the configuration data **130**, the

encryption circuit **125** is first initialized. In one embodiment of the invention, the upper flip-flops **240** are loaded with the initialization data **202** by asserting the select signal **203**. The initialization data **202** defines the starting state of the upper flip-flops **240**. Also as part of the initialization procedure, the key **180** is received and shifted into the key flip-flops **224**. In one embodiment of the invention, the upper flip-flops **240** are set during the initialization procedure. The lower flip-flops **230** are reset during the initialization. In another embodiment, the lower flip-flops **230** and the upper flip-flops **240** are set to a predefined pattern of 1's and 0's.

After the initialization procedure, the encryption procedure then begins generating the encrypted configuration data **135**. The key switch **220** output and the portion of the encrypted configuration data **135** stored in the lower flip-flops **230** are AND'ed in the AND gates **250**. The output of the AND gates **250** is then XOR'd to generate a signal X**204**. Each new configuration data **130** bit is XOR'd with the signal X**204** to generate a corresponding new encrypted configuration data **135** bit. The new encrypted configuration data **135** bit is shifted into the first flip-flop in the lower flip-flops **230**.

The upper flip-flops **240** shift bits from the first flip-flop to the last flip-flop. The outputs of the upper flip-flops **240** determine the value fed back into the first flip-flop. The outputs are also used as the inputs to the switch muxes **222**. Each switch mux **222** has two inputs from two different flip-flops in the upper flip-flops **240**. Each switch mux **222** has a select line connected to one of the key flip-flops **224**. Thus, the 1's and 0's in the key flip-flops **240** determine how the outputs of the upper flip-flops **240** are connected to the AND gates **250**. A change in the key **180** value effectively changes the connections to the AND gates **250**.

Table 1 illustrates an example set of encrypted configuration data **135** generated from the configuration data **130**. In this example, there are three upper flip-flops **240**, three lower flip-flops **230**. The key is one bit long and connects the outputs of the middle flip-flops in the upper and lower flip-flops to the middle AND gate, connects the outputs of the last flip-flop in the upper flip-flops **240** to the same AND gate as the first flip-flop in the lower flip-flops **230**, and vice-versa. Also, only the outputs of the last two upper flip-flops **240** are used as feedback to the first flip-flop. D is a bit in the configuration data **130**. D* is the corresponding bit in the encrypted configuration data **135**.

TABLE 1

Upper Flip-Flops 240			Lower Flip-Flops 230			AND Gates 250					
FF2	FF1	FF0	FF2	FF1	FF0	A2	A1	A0	X	D	D*
1	1	1	0	0	0	0	0	0	0	1	1
1	1	0	0	0	1	0	0	1	1	0	1
1	0	0	0	1	1	0	0	1	1	1	0
0	0	1	1	1	0	1	0	0	1	1	0
0	1	0	1	0	0	0	0	0	0	0	0
1	0	1	0	0	1	0	0	1	1	0	1
0	1	1	0	1	1	0	1	0	1	0	1
1	1	1	1	1	1	1	1	1	0	0	0

In one embodiment of the invention, the encryption circuit **125** does not include the upper flip-flops **240**. In this embodiment, the key switch **220** is connected to the outputs of the lower flip-flops **230**. Similar changes are made to the decryption circuit **115**.

In another embodiment of the invention, the initialization data **202** is received at the input to a load mux **290**. The other input to the load mux **290** is the encrypted configuration data **135** (the encrypted configuration data **135** is no longer connected directly to the first flip-flop in the lower flip-flops **230**). The output of the load mux **290** is connected to the input of the first flip-flop in the lower flip-flops **230**. The select signal **203** selects between the encrypted configuration data **135** or the initialization data **202**. The initialization signal **202** is no longer connected to the load mux **291**. The output of the last flip-flop in the lower flip-flops **230** is connected to the load mux **291** instead. Thus, the lower flip-flops **230** and the upper flip-flops **240** act as one long shift register when the select **203** signal is appropriately asserted. Thus, in this embodiment, the upper and lower flip-flops are loaded together during the initialization process.

In an embodiment of the invention where the security initialization circuit **114** does not transmit the key **180** to the storage device **120**, the encryption circuit **125** operates as follows. The key switch **220** receives the key **180** from the storage area. In one embodiment, the key flip-flops **224** are replaced with the storage area devices. In any case, the key switch **220** includes the key **180**. The select signal **203** is asserted to cause the upper and lower flip-flops to act as one long shift register. This long shift register is then loaded with the initialization **202** data. In one embodiment, the initialization data **202** is received from the PLD **110** (FIG. **1**).

Decryption Circuit

FIG. **3** illustrates a decryption circuit used in a programmable logic device. The decryption circuit **115** of FIG. **3** uses a relatively small number of gates and provides adequate protection of the circuit design as implemented in the PLD.

The decryption circuit **115** is very similar to the encryption circuit **125**. The similarity helps reduce the cost of designing the encryption and decryption circuits. (Note that the decryption circuit **115** is similar to the alternate embodiment having the load mux **290**.) The following describes the differences between the two circuits. The element in the decryption circuit **115** not included in the encryption circuit **125** is the load multiplexer **390**. The initialization signal **202** input of the load mux **291** has been changed to be the output of the last flip-flop in the lower flip-flops **230**. The load mux **390** has one input connected to the encrypted configuration data **135** and the other input connected to the initialization data **202**. Additionally, the XOR **260** is relabeled as XOR **360** to reflect that the operation being performed by the XOR **360** is different from the XOR **260** (i.e. $D^* \oplus X = D$ instead of $D \oplus X = D^*$).

The pattern of the connections that connect the inputs of the switch muxes **222** to the outputs of the upper flip-flops **240** are the same as in the encryption circuit **125**. Similarly, the connections to the XOR **249** from the upper flip-flops **240** must also be the same as in the encryption circuit **125**. If these two conditions are not true, then the value of X**204** may be different in the encryption circuit **125** than in the decryption circuit **115**, resulting in a failure of the encryption and decryption scheme.

Importantly, as will be shown below, the addition of the load mux **390** and the change to one of the inputs of the load mux **291**, allow the lower flip-flops **230** and the upper flip-flops **240** to act as one long shift register. By asserting the select signal **203**, the output of the last flip-flop in the lower flip-flops **230** is fed to the input of the first flip-flop of the upper flip-flops **240**. Thus, the initialization signal **202** can load all the bits in the both the upper and the lower flip-flops.

The following describes the operation of the decryption circuit **115**. The decryption circuit **115** supports an initial-

ization procedure and a decryption procedure. The initialization procedure causes the upper flip-flops **240** and the lower flip-flops **230** to be loaded with the values of the initialization signal **202**. In another embodiment, the initialization procedure simply resets the lower flip-flops **230** and sets the upper flip-flops **240**. In another embodiment, the lower flip-flops **230** and the upper flip-flops **240** are set to predetermined pattern of 1's and 0's. The key **180** is also loaded into the key flip-flops **224**. Importantly, the initial states of the key switch **220**, the upper flip-flops **240** and lower flip-flops **230** in the encryption circuit **125** must be the same as the initial states of the key switch **220**, the upper-flip-flops **240** and the lower flip-flops **230** in the decryption circuit **115**. Otherwise, the decryption circuit **115** will not be able to decrypt the encrypted configuration data **135**. During the decryption procedure, the encrypted configuration data **135** is received by the decryption circuit **115** and is XOR'd with the signal **X204**. The result of XOR'ing **X204** and the encrypted configuration data **135** is the configuration data **130**. The encrypted configuration data **135** is shifted through the lower flip-flops **230** to regenerate the same **X204** as was generated in the encryption circuit **125**.

Table 2 provides an example of decrypting the encrypted configuration data **135**. The same set of conditions used to generate Table 1 are used to generate Table 2. Importantly, the configuration data D of Table 2 is the same as the configuration data D of Table 1.

TABLE 2

Upper Flip-Flops 240			Lower Flip-Flops 230			AND Gates 250					
FF2	FF1	FF0	FF2	FF1	FF0	A2	A1	A0	X	D*	D
1	1	1	0	0	0	0	0	0	0	1	1
1	1	0	0	0	1	0	0	1	1	1	0
1	0	0	0	1	1	0	0	1	1	0	1
0	0	1	1	1	0	1	0	0	1	0	1
0	1	0	1	0	0	0	0	0	0	1	1
1	0	1	0	0	1	0	0	1	1	1	0
0	1	1	0	1	1	0	1	0	1	1	0
1	1	1	1	1	1	1	1	1	0	0	0

Daisy-Chained Programmable Logic Devices with Secure Configuration

FIG. **4** illustrates a number of daisy-chained programmable logic devices having security circuits. In one embodiment, the PLDs **110** support various configuration modes. Some of the configuration modes allow PLDs to be linked (daisy-chained) during the configuration process. One embodiment of the invention supports secure communications of the configuration data for daisy-chained PLDs.

The elements of FIG. **4** are the storage device **120**, a master PLD **410** and a slave PLD **420**. In one embodiment, the storage device **120**, the master PLD **410** and the slave PLD **420** communicate the key **180** and the encrypted configuration data **135** over the secure communications link **425**. (The secure communications link **425** includes a bus for communicating the key **180** and the encrypted configuration data **135**.) In this embodiment, each PLD is directly connected to the secure communications link **425** (e.g., master PLD **410** and slave PLD **420** are both connected to the secure communications link **425**). In another embodiment, the PLDs **110** are daisy-chained using the alternative secure communications link **435**. For example, the output from the encryption circuit **125** is fed to the input of the security circuit **111** of the master PLD **410**. After being programmed, the master PLD **410** feeds the remaining encrypted configuration data **135**, and the key **180**, to the next PLD in the daisy-chain (e.g., slave PLD **420**), using the alternative secure communications link **435**.

Prior to describing the use of the two secure communications links, some of configuration modes supported by the PLD are described. One embodiment supports a multiple PLD configuration mode where multiple PLDs **110** are all connected to the storage device **120** and are programmed one after the other. One embodiment supports secure storage of the PLD during configuration modes similar to the following configuration modes of the Xilinx, Inc. XC4000™ FPGA product family: a slave serial mode, a master serial mode, a master parallel up mode, and a master parallel down mode. The 1994 Xilinx Data Book, page 2–25 through page 2–46, describes the unsecured configuration modes for the XC4000™ FPGA product family. Another embodiment supports other secure configuration modes, such as, a secure version of the express mode (described in the XC5200™ FPGA data sheet from Xilinx, Inc.). The following briefly describes each of these modes and the various uses of the two secure communications links.

The multiple PLD configuration mode allows multiple PLDs **110** to be programmed from one storage device **120** using the secure communications link **425**. Each PLD listens to the secure communications link **425** to receive the encrypted configuration data **135** and the key **180** (and any additional keys **180** if the storage device **120** changes the keys during the configuration process). During the initialization procedure, each PLD performs the initialization procedure described above. During the configuration procedure, each PLD decrypts the encrypted configuration data **135** to generate the configuration data **130**. However, only one PLD at a time uses the configuration data **130** to program that PLD's **110** configurable logic elements **118**. For example, the master PLD **410** will program its configurable logic elements **118**, using the generated configuration data **130**, until the configurable logic elements **118** are completely programmed. Then the next PLD in the daisy-chain (slave PLD **420**) will use the generated configuration data **130** to program the configurable logic elements **118** of that next PLD. In one embodiment, when completely programmed, a PLD notifies the next PLD **110** in the chain to begin using the configuration data **130**. In another embodiment, the storage device **120** notifies each PLD when that PLD should begin programming itself with the configuration data **130**.

The slave serial mode is used for PLDs **110** that depend on other PLDs **110** for their configuration data **130**. For slave serial mode, the slave PLD **420** accepts the encrypted configuration data **135** from a single bit wide data bus, in the secured communications link **425**, and retransmits the encrypted configuration data **135** to the next slave PLD when that slave PLD **420** is completely programmed. Prior to decrypting the encrypted configuration data **135**, the slave PLD **420** loads the initialization data **202** into the upper flip-flops **240** and the lower flip-flops **230** of the slave PLD **420**'s decryption circuit **115**. Similarly, the slave PLD **420** loads the key **180** into the key flip-flops **224** of the slave PLD **420**'s decryption circuit **115**.

The master serial mode is used for the master PLD **410** in a chain of PLDs **110**. The chain of PLDs **110** are configured one after another. Additionally, the master serial mode is used where only one PLD is configured. The master serial mode is similar to the slave serial mode except that in the master serial mode, the master PLD **410** drives a configuration clock signal to the slave PLDs **420** and to the storage device **120**. Once fully programmed, the security circuit **111** causes the encrypted configuration data **135** to bypass the

decryption circuit **114** of the master PLD **410** and to be retransmitted to the slave PLD **420** via the alternative secure communications link **435**. Prior to beginning to retransmit the encrypted configuration data **135**, the master PLD **410** unloads the data in the upper and the lower flip-flops via the initialization out signal **302** (FIG. 3). The initialization out signal **302** is connected to the initialization signal **202** of the decryption circuit **115** of the slave PLD **420**. Similarly, the master PLD **410** unloads the key **180** in the key flip-flops **224** via the key out **182** signal (FIG. 3). The key out **182** signal is connected to the input of the first key flip-flop **224** of the decryption circuit **115** of the slave PLD **420**. In the embodiment where the key **180** is not communicated between the storage device **120** and the PLD (e.g., the key **180** is stored in a non-volatile memory in the PLD and a non-volatile memory in the storage device **120** prior to being shipped to a customer), no key out **182** signal is generated. In this embodiment, the initialization out signal **302** only includes the current state of the upper flip-flops **240** and the lower flip-flops **230**.

The master parallel up mode is used where the storage device **120** includes a multiple bit wide output (e.g., byte-wide output). In one embodiment having parallel configuration data bits, the parallel configuration data is XOR'd with X**204** or is XOR'd with taps from the lower flip-flops **230**. The encryption circuit **125** and the decryption circuit **115** thus support encrypted configuration data output and inputs that are multiple bits wide. The master parallel up mode is otherwise similar to the master serial mode.

The master parallel down mode is similar to the master parallel up mode except that the master PLD **410** generates addresses for the storage device **120** starting from the maximum address (e.g., 0×FFFFF) and counts down, instead of starting at the lowest address (e.g., 0×00000) and counting up.

The express mode is used when fast configuration of multiple PLDs **110** is desired. The express mode is a parallel configuration mode for all the PLDs **110** in a chain of PLDs **110**. The encrypted configuration data **135** is clocked in from the storage device **120** synchronously. A signal from the first PLD in the chain is used by that PLD to tell the next PLD to begin reading the data inputs for its encrypted configuration data **135**.

Other embodiments of the invention support other secure configuration modes such as synchronous peripheral mode and asynchronous peripheral mode.

The above describes a method and apparatus for securing a circuit design as implemented in a programmable logic device. A secure communications link between a storage device and one or more PLDs is first established transmitting encrypted configuration data to the PLD from the storage device.

What is claimed is:

1. A method of communicating encrypted configuration data between a programmable logic device (PLD) and a storage device, the method comprising:

storing first and second unencrypted configuration data for the PLD in the storage device, the first configuration data for implementing a public key cryptography circuit, the second configuration data for implementing a desired circuit design;

transmitting the unencrypted first configuration data to the PLD,

configuring the PLD using the unencrypted first configuration data to implement the public key cryptography circuit;

establishing a secure communications link between the storage device and the PLD using the public key cryptography circuit, whereby the storage device and the PLD exchange a secret key;

encrypting the second configuration data using the secret key;

transmitting the encrypted configuration data from the storage device to the PLD; and

decrypting the encrypted configuration data using the secret key to generate the second configuration data.

2. The method of claim **1**, further comprising:

re-configuring the PLD using the second configuration data.

* * * * *

UNITED STATES PATENT AND TRADEMARK OFFICE

CERTIFICATE OF CORRECTION

PATENT NO. : 6,212,639 B1
DATED : April 3, 2001
INVENTOR(S) : Charles R. Erickson

It is certified that error appears in the above-identified patent and that said Letters Patent is hereby corrected as shown below:

Title page,
Item [75], Inventors, delete the following individuals as inventors: "**Danesh Tavana**, Mountain View California and **Victor A. Holen**, Los Gatos California"

Signed and Sealed this

Twenty-second Day of October, 2002

Attest:

JAMES E. ROGAN
Director of the United States Patent and Trademark Office

Attesting Officer

US006305005B1

(12) **United States Patent**
Burnham

(10) **Patent No.: US 6,305,005 B1**
(45) **Date of Patent: Oct. 16, 2001**

(54) **METHODS TO SECURELY CONFIGURE AN FPGA USING ENCRYPTED MACROS**

(75) Inventor: **James L. Burnham**, Morgan Hill, CA (US)

(73) Assignee: **Xilinx, Inc.**, San Jose, CA (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

(21) Appl. No.: **09/231,528**

(22) Filed: **Jan. 14, 1999**

(51) **Int. Cl.**[7] **G06F 17/50**; H04L 9/00

(52) **U.S. Cl.** **716/16**; 716/18; 713/1; 713/100; 713/202

(58) **Field of Search** 716/1–21; 713/1, 713/100, 202; 380/4

(56) **References Cited**

U.S. PATENT DOCUMENTS

Re. 34,363		8/1993	Freeman et al. .	
3,849,760		11/1974	Endou et al. .	
4,849,904	*	7/1989	Aipperspach et al.	716/17
5,084,636		1/1992	Yoneda .	
5,128,871	*	7/1992	Schmitz	716/17
5,197,016		3/1993	Sugimoto et al. .	
5,237,218		8/1993	Josephson et al. .	
5,237,219		8/1993	Cliff .	
5,343,406		8/1994	Freeman et al. .	
5,394,031		2/1995	Britton et al. .	
5,457,408		10/1995	Leung .	
5,574,655		11/1996	Knapp et al. .	
5,705,938		1/1998	Kean .	
5,909,658	*	6/1999	Clarke et al.	701/126
5,946,478	*	8/1999	Lawman	716/17
6,205,574		3/2001	Dellinger et al. .	

FOREIGN PATENT DOCUMENTS

A0253530	6/1987	(EP) .
WO92/20157	11/1992	(WO) .
WO94/10754	11/1993	(WO) .
WO94/01867	11/1994	(WO) .

OTHER PUBLICATIONS

Wong et al. ("A single–chip FPGA implementation of the data encryption standard (DES) algorithm", IEEE Global Telecommunications Conference, 1998, GLOBECOM 1998, The Bridge to Global Integration, vol. 2, pp 827–832), Nov. 1998.*

Runje et al. ("Universal strong encryption FPGA core implementation", Proceedings of Design, Automation and Test in Europe, 1998, pp. 923–924, Feb. 1998.*

Kean et al. ("DES key breaking, encryption and decryption on the XC6216", Proceedings of IEEE Symposium on FPGAs for Custom Computing Machines, 1998, pp. 310–311), Apr. 1998.*

"The Programmable Logic Data Book", published Sep., 1996, in its entirety and also specifically pp. 4–54 to 4–79 and 4–253 to 4–286, available from Xilinx, Inc., 2100 Logic Drive, San Jose, California 95124.

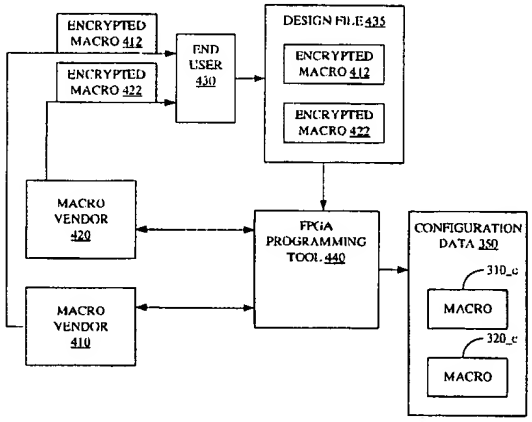
(List continued on next page.)

Primary Examiner—Matthew Smith
Assistant Examiner—Phallaka Kik
(74) *Attorney, Agent, or Firm*—Edward S. Mao

(57) **ABSTRACT**

A method is provided for securely configuring an FPGA with macros. Specifically, if an end user desires to use a macro from a macro vendor, the end user creates a design file containing an encrypted macro received from the macro vendor rather than the actual macro. The end user uses a FPGA programming tool to convert the design file into configuration data. Specifically, the FPGA programming tool processes the design file to detect encrypted macros. If an encrypted macro is detected, the FPGA programming tool requests authorization over a secured medium to decrypt the encrypted macro from the macro vendor. If authorization is received, the FPGA programming tool decrypts the encrypted macro and converts the design file into configuration data incorporating the macro.

17 Claims, 5 Drawing Sheets



OTHER PUBLICATIONS

"Core Solutions Data Book", published May, 1997, pp. 2–5 to 2–13 available from Xilinx, Inc. 2100 Logic Drive, San Jose, California 95124.

"The Programmable Logic Data Book", published 1994, available from Xilinx, Inc., 2100 Logic Drive, San Jose, California 95124, pp. 2–105 to 2–132 and 2–231 to 2–238.

D.D. Gajski, et al., "Computer Architecture" IEEE Tutorial Manual, IEEE Computer Society, 1987, pp. v–i.

"New IEEE Standard Dictionary of Electrical and Electronics Terms", Fifth Edition, 1993, p. 1011.

"IEEE Standard Test Access Port and Boundary–Scan Architecture", IEEE Std. 1149.1, published Oct. 21, 1993.

David A. Patterson and John L. Hennessy, "Computer Architecture: A Quantitive Approach", pp. 200–201, 1990.

Betty Prince, "Semiconductor Memories", copyright 1983, 1991, John Wiley & Sons, pp. 149–174.

Hodges, et al., "Analog MOS Integrated Circuits" IEEE Press, 1980, pp. 2–11.

"The Programmable Logic Data Book", published 1998, available from Xilinx, Inc., 2100 Logic Drive, San Jose, California 95124, pp. 4–46 to 4–59.

McCarley et al., "Macro–Instruction Generation for Dynamic Logic Caching", Proceedings of the 8th IEEE International Workshop on Rapid System Prototyping, Jun. 24–26, 1997, pp. 63–69.

Patriquin et al., "An Automated Design Process for the CHAMP Module", Proceedings of the IEEE 1995 National Aerospace and Electronics Conference, May 22–26, 1995, NAEC, vol. 1, pp. 417–424.

Box, "Field Programmable Gate Array Based Configurable Preprocessor", Proceedings of the IEEE 1994 National Aerospace and Electronics Conference, May 23–27, 1994, NAECON 1994, vol. 1, pp. 427–434.

Shi et al., "Macro Block Based FPGA Floorplanning", Proceedings of the Tenth International Conference on VLSI Design, Jan. 4–7, 1997, pp. 21–26.

"The Programmable Logic Data Book", published 1998, available from Xilinx, Inc., 2100 Logic Drive, San Jose, California 95124, pp. 4–46 to 4–59.

* cited by examiner

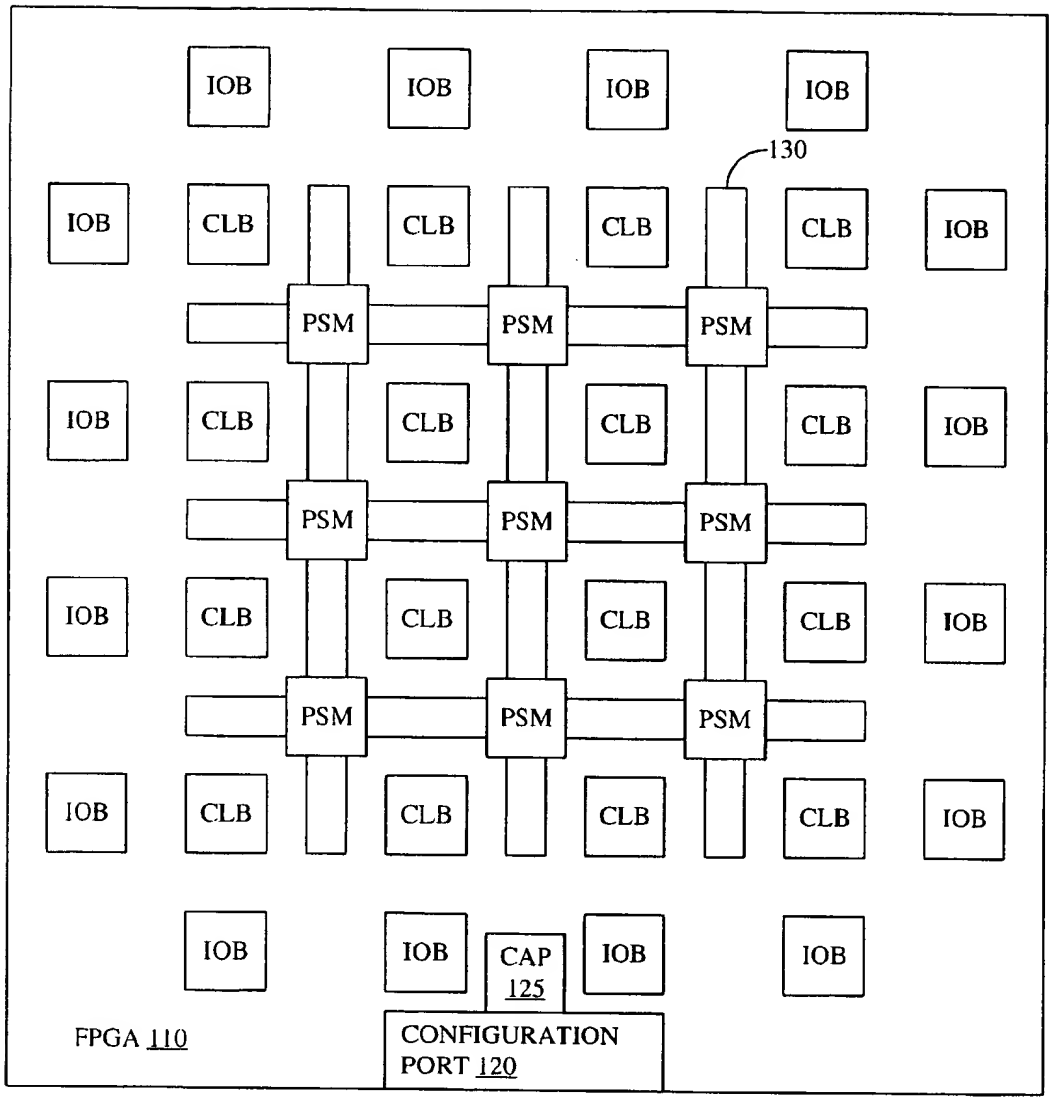


FIGURE 1 (Prior Art)

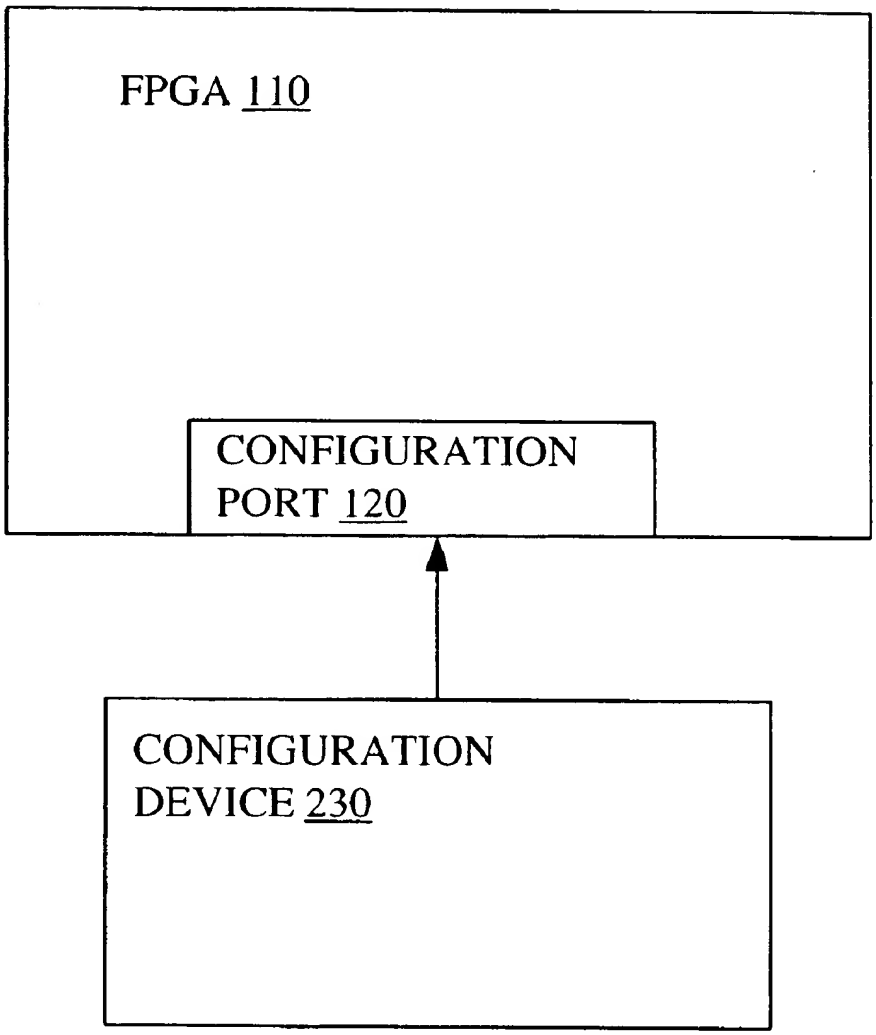


FIGURE 2 (Prior Art)

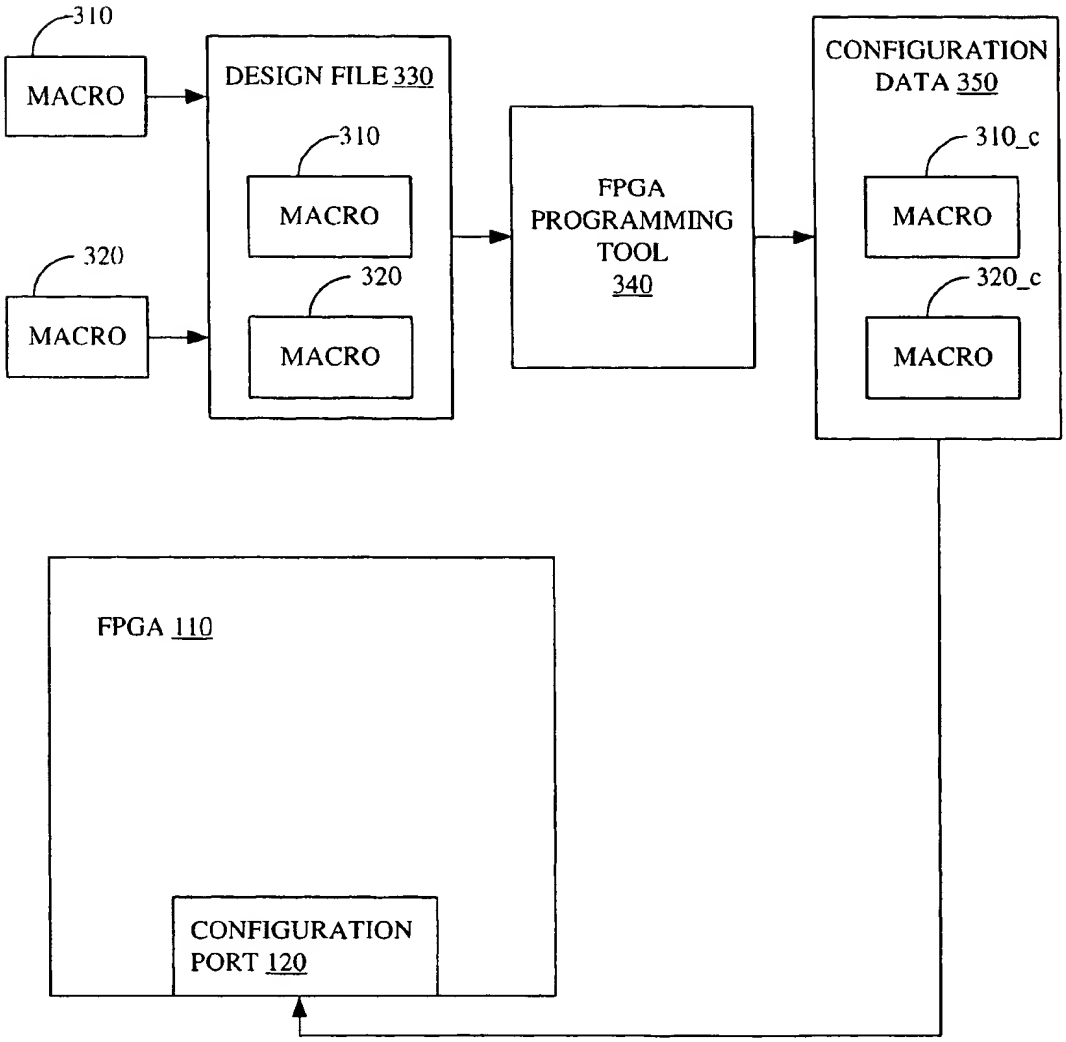


FIGURE 3 (Prior Art)

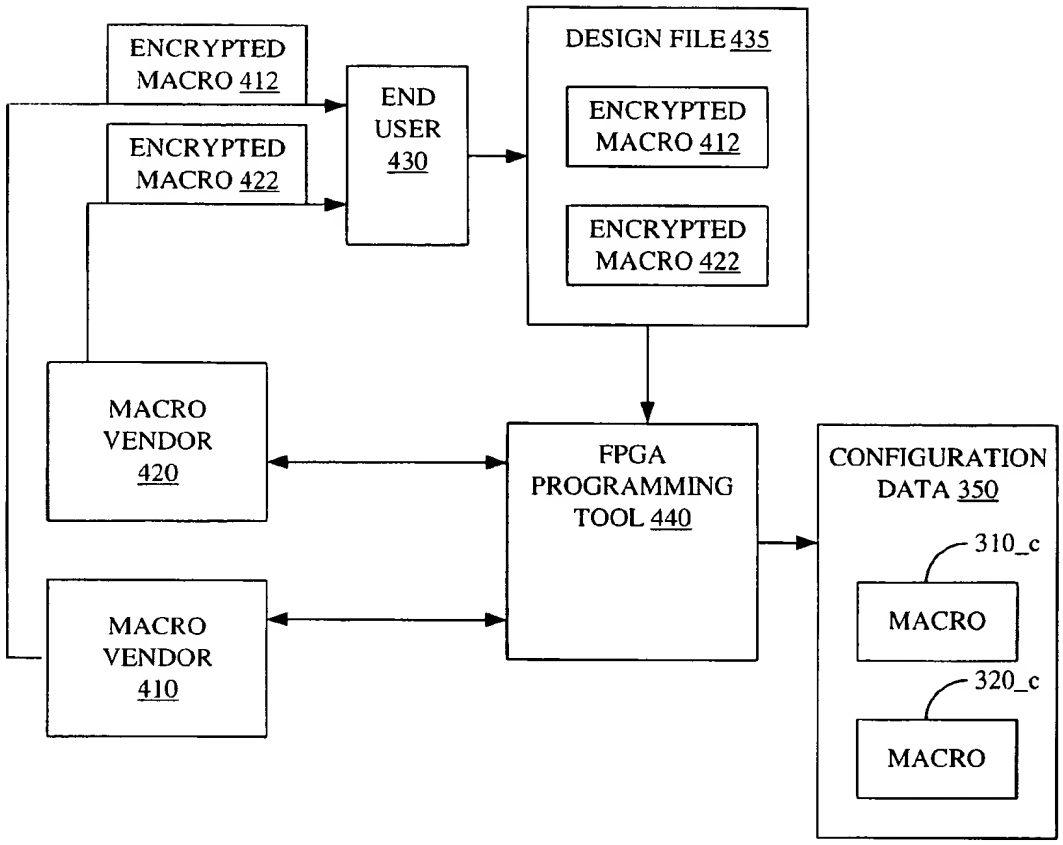

ENCRYPTED MACRO 412
ENCRYPTED MACRO 422
END USER 430
DESIGN FILE 435
ENCRYPTED MACRO 412
ENCRYPTED MACRO 422
MACRO VENDOR 420
MACRO VENDOR 410
FPGA PROGRAMMING TOOL 440
CONFIGURATION DATA 350
310_c
MACRO
320_c
MACRO

FIGURE 4

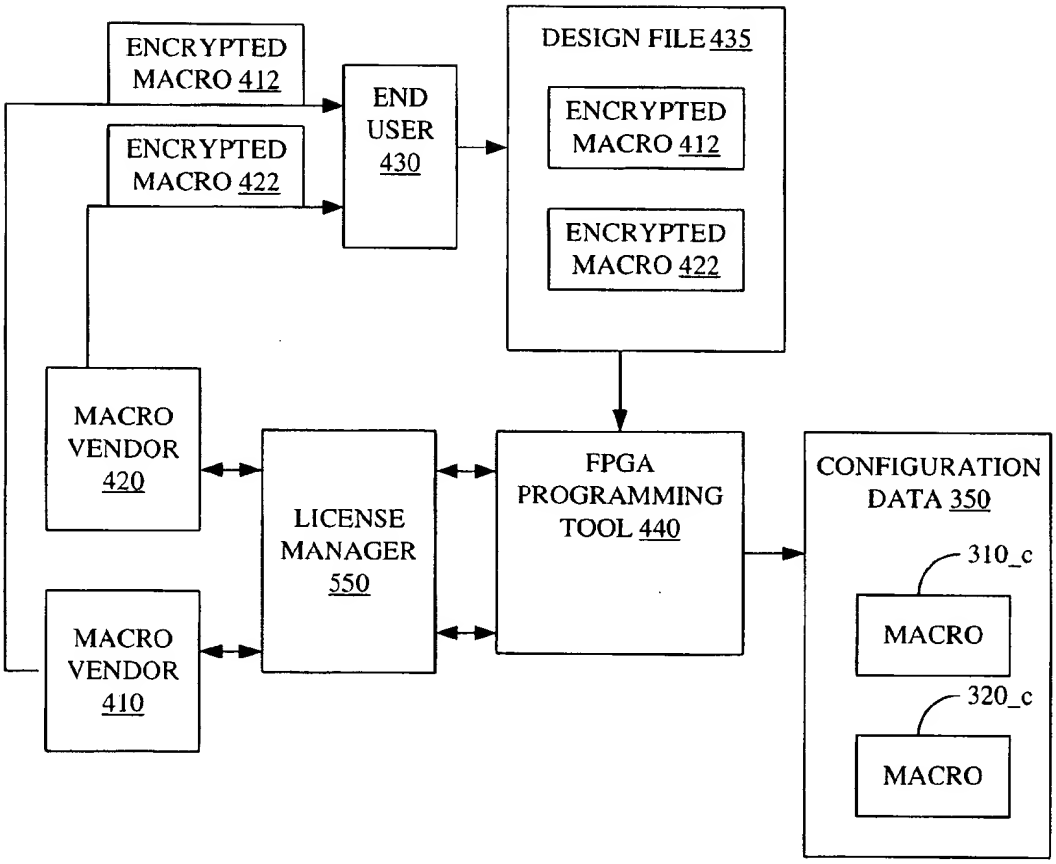

ENCRYPTED MACRO 412
ENCRYPTED MACRO 422
END USER 430
DESIGN FILE 435
ENCRYPTED MACRO 412
ENCRYPTED MACRO 422
MACRO VENDOR 420
MACRO VENDOR 410
LICENSE MANAGER 550
FPGA PROGRAMMING TOOL 440
CONFIGURATION DATA 350
310_c
MACRO
320_c
MACRO

FIGURE 5

METHODS TO SECURELY CONFIGURE AN FPGA USING ENCRYPTED MACROS

CROSS-REFERENCE TO RELATED APPLICATIONS

This application relates to concurrently filed, co-pending U.S. patent application Ser. No. 09/232,022, "FPGA CUSTOMIZABLE TO ACCEPT SELECTED MACROS", by Burnham et. al., owned by the assignee of this application and incorporated herein by reference.

This application relates to concurrently filed, co-pending U.S. patent application Ser. No. 09/232,021, "METHODS TO SECURELY CONFIGURE AN FPGA TO ACCEPT SELECTED MACROS", by Lawman et. al., owned by the assignee of this application and incorporated herein by reference.

This application relates to concurrently filed, co-pending U.S. patent application Ser. No. 09/231,912, "METHODS TO SECURELY CONFIGURE AN FPGA USING MACRO MARKERS", by Burnham et al., owned by the assignee of this application and incorporated herein by reference.

This application relates to U.S. patent application Ser. No. 09/000,519, entitled "DECODER STRUCTURE AND METHOD FOR FPGA CONFIGURATION" by Gary R. Lawman, which is also incorporated herein by reference.

BACKGROUND OF THE INVENTION

1. Field of the Invention

The present invention relates to programmable devices such as field programmable gate arrays (FPGAs). More specifically, the present invention relates to methods for programming licensed software in an FPGA.

2. Discussion of Related Art

Due to advancing semiconductor processing technology, integrated circuits have greatly increased in functionality and complexity. For example, programmable devices such as field programmable gate arrays (FPGAs) and programmable logic devices (PLDs), can incorporate ever-increasing numbers of functional blocks and more flexible interconnect structures to provide greater functionality and flexibility.

FIG. **1** is a simplified schematic diagram of a conventional FPGA **110**. FPGA **110** includes user logic circuits such as input/output blocks (IOBs), configurable logic blocks (CLBs), and programmable interconnect **130**, which contains programmable switch matrices (PSMs). Each IOB and CLB can be configured through configuration port **120** to perform a variety of functions. Programmable interconnect **130** can be configured to provide electrical connections between the various CLBs and IOBs by configuring the PSMs and other programmable interconnection points (PIPS, not shown) through configuration port **120**. Typically, the IOBs can be configured to drive output signals or to receive input signals from various pins (not shown) of FPGA **110**.

FPGA **110** also includes dedicated internal logic. Dedicated internal logic performs specific functions and can only be minimally configured by a user. For example, configuration port **120** is one example of dedicated internal logic. Other examples may include dedicated clock nets (not shown), power distribution grids (not shown), and boundary scan logic (i.e. IEEE Boundary Scan Standard 1149.1, not shown).

FPGA **110** is illustrated with 16 CLBs, 16 IOBs, and 9 PSMs for clarity only. Actual FPGAs may contain thousands of CLBs, thousands of IOBs, and thousands of PSMs. The ratio of the number of CLBs, IOBs, and PSMs can also vary.

FPGA **110** also includes dedicated configuration logic circuits to program the user logic circuits. Specifically, each CLB, IOB, PSM, and PIP contains a configuration memory (not shown) which must be configured before each CLB, IOB, PSM, or PIP can perform a specified function. Typically the configuration memories within an FPGA use static random access memory (SRAM) cells. The configuration memories of FPGA **110** are connected by a configuration structure (not shown) to configuration port **120** through a configuration access port (CAP) **125**. A configuration port (a set of pins used during the configuration process) provides an interface for external configuration devices to program the FPGA. The configuration memories are typically arranged in rows and columns. The columns are loaded from a frame register which is in turn sequentially loaded from one or more sequential bitstreams. (The frame register is part of the configuration structure referenced above.) In FPGA **110**, configuration access port **125** is essentially a bus access point that provides access from configuration port **120** to the configuration structure of FPGA **110**.

FIG. **2** illustrates a conventional method used to configure FPGA **110**. Specifically, FPGA **110** is coupled to a configuration device **230** such as a serial programmable read only memory (SPROM), an electrically programmable read only memory (EPROM), or a microprocessor. Configuration port **120** receives configuration data, usually in the form of a configuration bitstream, from configuration device **230**. Typically, configuration port **120** contains a set of mode pins, a clock pin and a configuration data input pin. Configuration data from configuration device **230** is transferred serially to FPGA **110** through the configuration data input pin. In some embodiments of FPGA **110**, configuration port **120** comprises a set of configuration data input pins to increase the data transfer rate between configuration device **230** and FPGA **110** by transferring data in parallel. However, due to the limited number of dedicated function pins available on an FPGA, configuration port **120** usually has no more than eight configuration data input pins. Further, some FPGAs allow configuration through a boundary scan chain. Specific examples for configuring various FPGAs can be found on pages 4–46 to 4–59 of "The Programmable Logic Data Book", published in January, 1998 by Xilinx, Inc., and available from Xilinx, Inc., 2100 Logic Drive, San Jose, Calif. 95124, which pages are incorporated herein by reference. Additional methods to program FPGA's are described by Lawman in U.S. patent application Ser. No. 09/000,519, entitled "DECODER STRUCTURE AND METHOD FOR FPGA CONFIGURATION" by Gary R. Lawman, which is referenced above.

As explained above, actual FPGAs can have thousands of CLBs, IOBs, PSMs, and PIPs; therefore, the design and development of FPGA software is very time-consuming and expensive. Consequently, many vendors provide macros for various functions that can be incorporated by an end user of the FPGA into the user's own design file. For example, Xilinx, Inc. provides a PCI interface macro, which can be incorporated by an end user into the user's design file. The user benefits from the macro because the user does not need to spend the time or resources to develop the macro. Further, since the vendor profits from selling the same macro to many users, the vendor can spend the time and resources to design optimized macros. For example, the vendor strives to provide macros having high performance, flexibility, and low gate count. However, the macro vendors are reluctant to give

out copies of the macros without a way of insuring that the macros are used only by licensed users. Hence, there is a need for a method or structure to insure third party macros are used only by licensed end users.

SUMMARY

The present invention uses encrypted macros in design files to insure only licensed users can use specific macros in FPGAs. Specifically, an end user creates a design file by incorporating an encrypted macro in the end user's FPGA design file instead of the actual source code for the macro. The end user uses an FPGA programming tool that is configured to request authorization from the macro vendor or a license manager to decrypt the encrypted macro. If authorization is received, the FPGA programming tool converts the design file into configuration data, which incorporates the macro.

The macro vendor only provides encrypted macros to the end user. The encrypted macros cannot be used in an FPGA unless the FPGA programming tool obtains authorization to decrypt the encrypted macro. To obtain authorization, the FPGA programming tool contacts the macro vendor over a secure medium such as a telephone line or a secure channel of a network. In one embodiment, the macro vendor supplies a decryption key that is needed to decrypt the encrypted macro over the secured medium to the FPGA programming tool. Furthermore, in some embodiments, the end user must provide a unique and confidential user identification to the FPGA programming tool. The user identification is provided to the macro vendor when the FPGA programming tool requests authorization to decrypt the encrypted macro. Thus, the macro vendor can determine whether the end user is licensed to use the encrypted macro. Moreover, the end user can include several encrypted macros in a single design file. The FPGA programming tool is configured to detect the various encrypted macros and to contact the appropriate macro vendor to obtain authorization for each encrypted macro. The present invention will be more fully understood in view of the following description and drawings.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. **1** is a simplified schematic diagram of a conventional FPGA.

FIG. **2** is a prior art schematic diagram of an FPGA configured with a configuration device.

FIG. **3** illustrates a prior art method of programming an FPGA using macros.

FIG. **4** illustrates a method of programming an FPGA in accordance with one embodiment of the present invention.

FIG. **5** illustrates a method of programming an FPGA in accordance with one embodiment of the present invention.

DETAILED DESCRIPTION

FIG. **3** illustrates a conventional manner of configuring an FPGA using macros. In FIG. **3**, an end user (not shown) desires to program FPGA **110** with a macro **310** and a macro **320**. Typically, the end user creates a design file **330**, which includes macro **310** and macro **320**. Design file **330** is converted into configuration data **350** by an FPGA programming tool **340**. Configuration data **350** contains macro **310**_c, which is the converted version of macro **310**, and macro **320**_c, which is the converted version of macro **320**. The converted versions of the macros typically take the form of portions of a configuration bitstream. Configuration data **350** also typically takes the form of a configuration bitstream. Configuration data **350** is typically stored in a configuration device such as configuration device **230** (FIG. **2**). Configuration data **350** is sent into FPGA **110** through configuration port **120** to configure FPGA **110**. As stated above, the macro vendors providing macros **310** and **320** may not wish to make macro **310** and macro **320** easily available due to fear of unlicensed use.

FIG. **4** illustrates a method to program macros into FPGAs without fear of piracy in accordance with one embodiment of the present invention. In FIG. **4**, an end user **430** wishes to configure an FPGA using macro **310** from macro vendor **410** and macro **320** from a macro vendor **420**. Instead of providing a copy of macro **310** to end user **430**, macro vendor **410** provides an encrypted macro **412** to end user **430**. Similarly, macro vendor **420** provides a encrypted macro **422** to end user **430** instead of providing macro **320**. The specific encryption schemes used for encrypted macros **412** and **422** are not an integral part of the present invention. In one embodiment of the present invention, encrypted macros **412** and **422** are Java™ archive files. ("Java" is a trademark of Sun Microsystems, Inc.)

Encrypted macro **412** contains information regarding macro **310** that can be used by end user **430** to create and test a design file incorporating macro **310**. Specifically, encrypted macro **412** contains information regarding the FPGA resources (e.g., CLBs, IOBs, PSMs) required by macro **310**. Furthermore, encrypted macro **412** may contain information regarding how circuits designed by end user **430** can interface to the circuits defined by macro **310**. In some embodiments, encrypted macro **412** also contains information for simulating the circuits defined by macro **310** so that end user **430** can simulate the design file containing encrypted macro **412**. Encrypted macro **422** may contain similar information regarding macro **320**.

End user **430** creates a design file **435** containing encrypted macro **412** and encrypted macro **422**. End user **430** then uses an FPGA programming tool **440** to convert design file **435** into configuration data **350**, which contains macro **310**_c and macro **320**_c as described above. During the conversion of design file **435** into configuration data **350**, FPGA programming tool **440** processes design file **435** to locate encrypted macros. When an encrypted macro is detected, FPGA programming tool **440** contacts the macro vendor that created the encrypted macro over a secure medium, such as a telephone line or a secured channel of a public network, to obtain authorization to decrypt the encrypted macro. In one embodiment, FPGA programming tool **440** is a Java applet, which communicates with macro vendors using encrypted channels over the internet.

In the example of FIG. **4**, FPGA programming tool **440** contacts macro vendor **420** to obtain authorization to decrypt encrypted macro **422**. Similarly, FPGA programming tool **440** contacts macro vendor **410** to obtain authorization to decrypt encrypted macro **412**. In some embodiments, the macro vendors provide a decryption key to FPGA programming tool **440** for decrypting the encrypted macro. FPGA programming tool **440** is configured so that the decrypted macro is not revealed to end user **430**. In one embodiment, FPGA programming tool **440** is configured to program exactly one FPGA or exactly one configuration device each time authorization is given to decrypt a macro.

Each end user (or the company employing the end user) has a unique and confidential password to identify the end user. When FPGA programming tool **440** requests authorization for encrypted macro **412** from macro vendor **410**, FPGA programming tool **440** transmits the password of end

user **430** to macro vendor **410**. Thus, macro vendor **410** is provided with the identity of the end user requesting use of encrypted macro **412**. Thus, macro vendor **410** can determine whether the request to decrypt encrypted macro **412** comes from a licensed end user. Furthermore, macro vendor **410** can track how often encrypted macro **412** is used. Similarly, macro vendor **420** is able track the use of encrypted macro **422**. Thus, encrypted macros **412** and **422** can be freely distributed with only minimal risk of unauthorized use.

FIG. **5** illustrates a method to program macros into FPGAs without fear of piracy in accordance with another embodiment of the present invention. Since the method of FIG. **5** is similar to the method of FIG. **4**, only the differences between the methods are described. Specifically, for the method of FIG. **5**, FPGA programming tool **440** contacts a license manager **550** rather than macro vendors **410** and **420**. License manager **550** provides authorization for FPGA programming tool **440** to decrypt encrypted macros **422** and **412**. License manager **550** manages the authorization of encrypted macros for a variety of macro vendors, such as macro vendors **410** and **420**. Thus, each end user pays licensing fees to and requests passwords from license manager **550** to use the encrypted macros. License manager **550** distributes the licensing fees to the appropriate macro vendors. Thus, encrypted macros **412** and **422** can be freely distributed with only minimal risk of unauthorized use.

In the various embodiments of this invention, methods and structures have been described to securely distribute and use third party macros. Only encrypted versions of the macros are released to end users. To use the encrypted macros an end user must receive authorization or a decryption key from the macro vendor to convert a design file using the encrypted macro into configuration data. Thus, the possibility of unlicensed use of the macro is diminished. By providing a secure method to distribute macros, macro vendors are motivated to expend the time and effort to create large libraries of optimized macros to sell to end users. Thus, the cost and time for creating design files for FPGAs by an end user can be reduced through the use of macros from macro vendors.

The various embodiments of the structures and methods of this invention that are described above are illustrative only of the principles of this invention and are not intended to limit the scope of the invention to the particular embodiments described. For example, in view of this disclosure, those skilled in the art can define other design files, encrypted macros, macros, encryption algorithms, configuration devices, programming tools, FPGAs, CLBs, IOBs, PSMs, configuration access ports, configuration ports, and so forth, and use these alternative features to create a method, circuit, or system according to the principles of this invention. Thus, the invention is limited only by the following claims.

What is claimed is:

1. A method to program an FPGA comprising the steps of:

creating a design file incorporating a first encrypted macro of a first macro;

obtaining authorization to decrypt said first encrypted macro;

decrypting the first encrypted macro to obtain the first macro; and

converting said design file and said first macro into configuration data, wherein said configuration data incorporates a first converted macro based on said first macro.

2. The method of claim **1**, further comprising the step of obtaining said first encrypted macro from a first macro vendor.

3. The method of claim **1**, wherein said step of obtaining authorization to decrypt said first encrypted macro further comprises the step of obtaining a first decryption key for said first encrypted macro.

4. The method of claim **3**, wherein said authorization is obtained over a telephone line.

5. The method of claim **3**, wherein said authorization is obtained over a secured channel of a network.

6. The method of claim **1**, further comprising the step of programming said FPGA with said configuration data.

7. The method of claim **1**, further comprising the step of incorporating a second encrypted macro of a second macro in said design file.

8. The method of claim **7**, wherein said configuration data incorporates a second converted macro based on said second macro.

9. The method of claim **7**, further comprising the step of obtaining authorization to decrypt said second encrypted macro.

10. The method of claim **9**, further comprising the step of obtaining said second encrypted macro from a second macro vendor.

11. The method of claim **1**, wherein said first encrypted macro is a Java archive file.

12. The method of claim **1**, further comprising the step of providing a confidential password for obtaining authorization to decrypt said first encrypted macro.

13. A method to securely provide macros for programming an FPGA, said method comprising the steps of:

encrypting a first macro to form a first encrypted macro;

providing said first encrypted macro to an end user;

granting authorization to decrypt said first encrypted macro over a secured medium; and

converting the first macro into a first converted macro.

14. The method of claim **13**, further comprising the step of providing a first decryption key to decrypt said first encrypted macro over said secured medium.

15. The method of claim **13**, wherein said secured medium is a telephone line.

16. The method of claim **13**, wherein said secured medium is a secured channel of a network.

17. The method of claim **13**, further comprising the steps of:

encrypting a second macro to form a second encrypted macro;

providing said second encrypted macro to an end user; and

granting authorization to decrypt said second encrypted macro over said secured medium.

* * * * *

US005530753A

United States Patent [19]

Easter et al.

[11] Patent Number: 5,530,753

[45] Date of Patent: Jun. 25, 1996

[54] **METHODS AND APPARATUS FOR SECURE HARDWARE CONFIGURATION**

[75] Inventors: **Randall J. Easter; Vincent A. Spano; Myron W. Zajac; John E. Campbell,** all of Dotchess County, N.Y.

[73] Assignee: **International Business Machines Corporation,** Armonk, N.Y.

[21] Appl. No.: **290,697**

[22] Filed: **Aug. 15, 1994**

[51] **Int. Cl.**[6] **H04K 1/00**

[52] **U.S. Cl.** **380/4**; 380/23

[58] **Field of Search** 380/3, 4, 23

[56] **References Cited**

U.S. PATENT DOCUMENTS

4,670,857	6/1987	Rackman .	
4,817,140	3/1989	Chandra et al.	380/4
4,872,140	10/1989	Graham et al. .	
4,941,174	7/1990	Ingham .	
5,148,534	9/1992	Comerford	380/4
5,247,577	9/1993	Bailey et al. .	
5,293,424	3/1994	Holtey et al. .	
5,301,143	4/1994	Ohri et al. .	
5,321,840	6/1994	Ahlin et al. .	

FOREIGN PATENT DOCUMENTS

9209465 5/1993 WIPO .

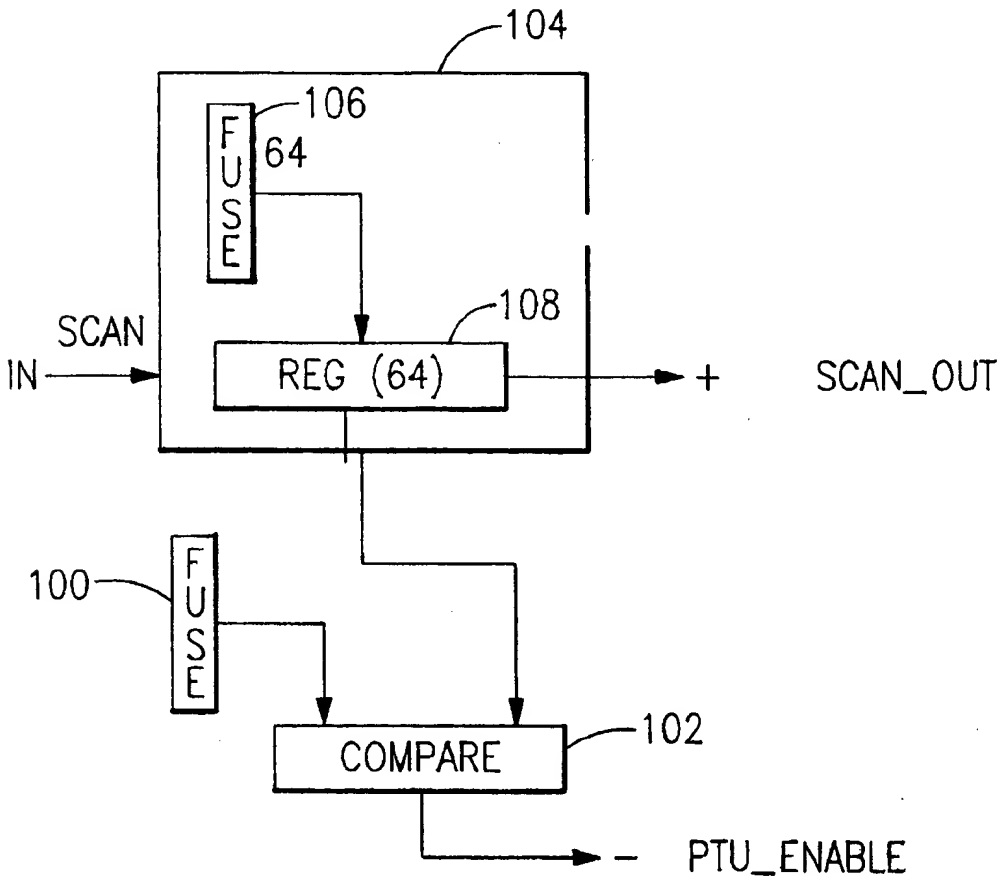
Primary Examiner—David C. Cain

Attorney, Agent, or Firm—Lynn L. Augspurger; C. Lamont Whitham

[57] **ABSTRACT**

Methods and apparatus are provided for electronically configuring hardware features and options. A computer chip encoding method is provided in which a predetermined code or encryption sequence is uniquely associated with a computer chip. This code is used to modify a hardware configuration by enabling new features or options. The systems and methods reduce manufacturing and inventory costs by allowing a generic product to be produced which is then customized to meet the needs of the user. In addition, features and options of a data processing system can be dynamically upgraded without interruption of service or hardware replacement.

12 Claims, 4 Drawing Sheets



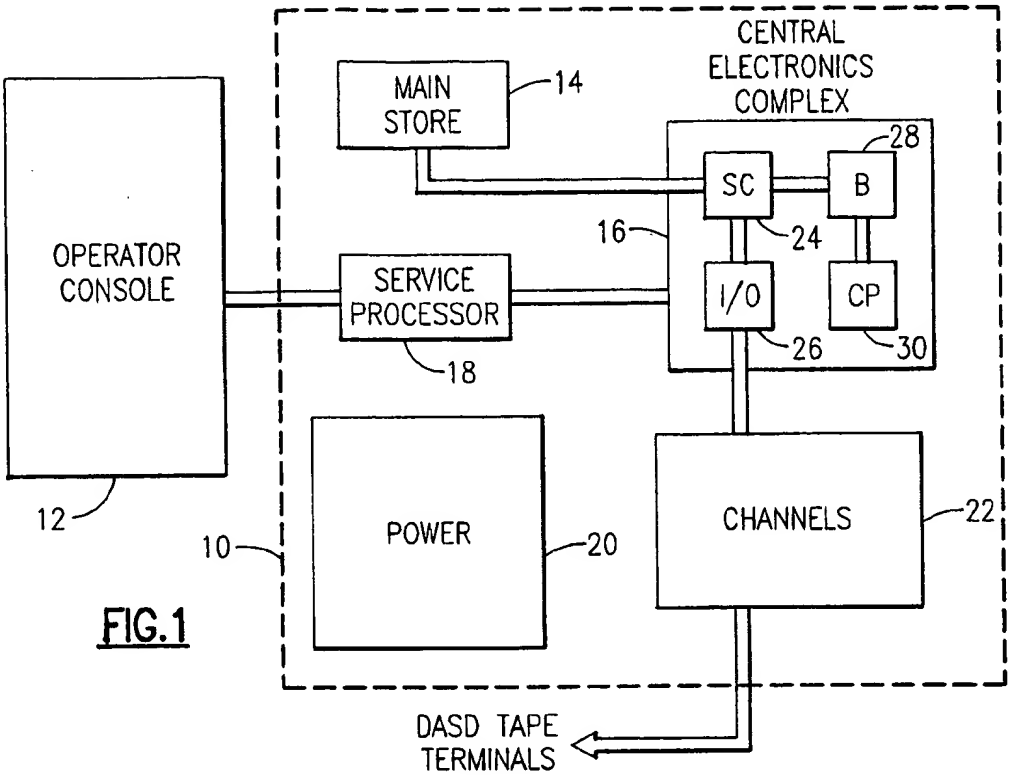

CENTRAL
ELECTRONICS
COMPLEX
MAIN
STORE
14
28
SC
B
16
24
OPERATOR
CONSOLE
SERVICE
PROCESSOR
I/O
CP
26
30
18
12
10
POWER
20
CHANNELS
22
DASD TAPE
TERMINALS

FIG.1

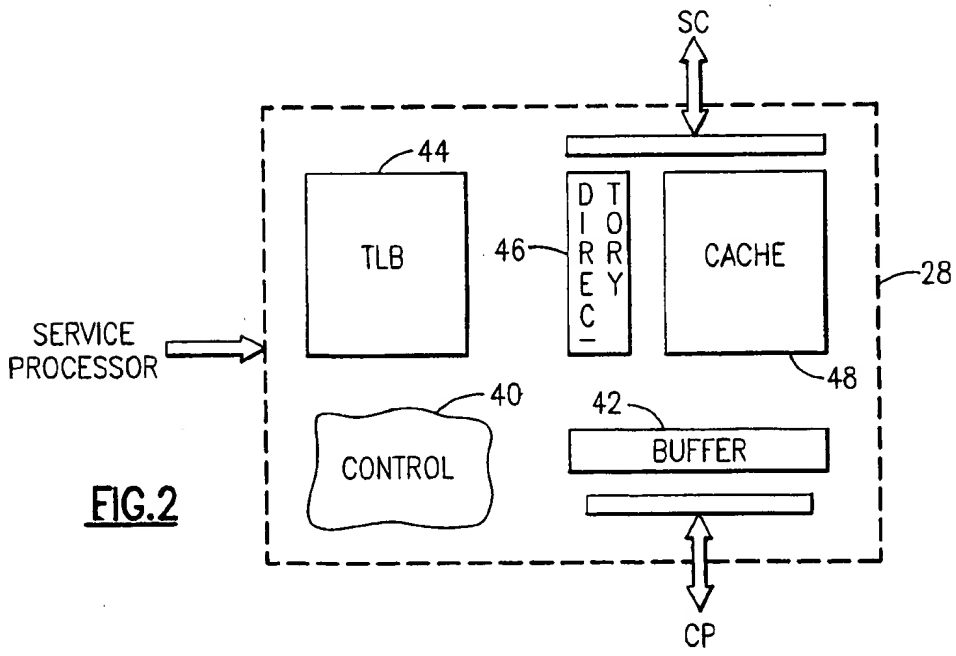

SC
44
TLB
46
DIRECTORY
CACHE
28
SERVICE
PROCESSOR
48
40
42
CONTROL
BUFFER
CP

FIG.2

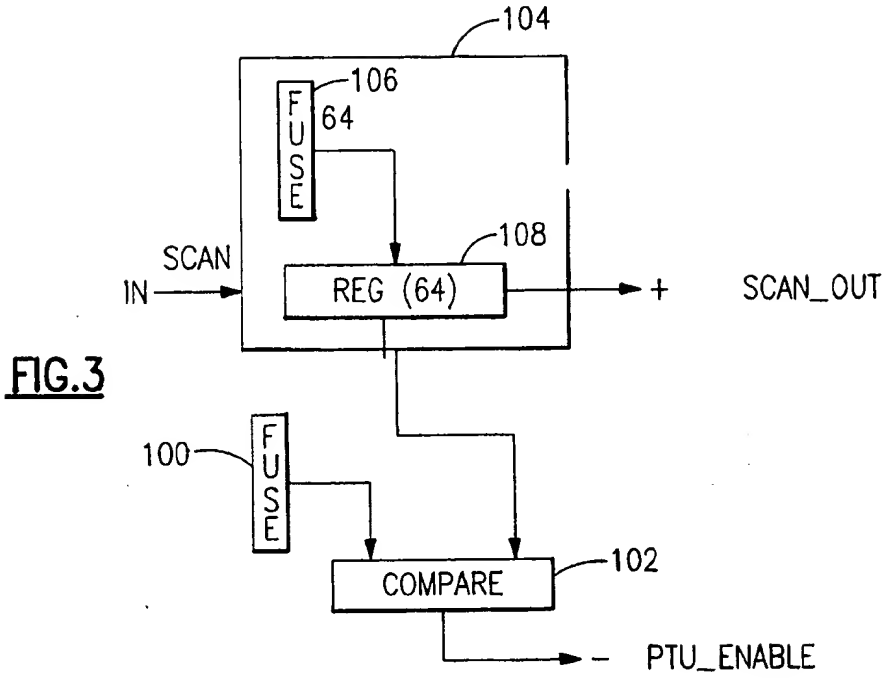

104
106
FUSE
64
108
SCAN
IN
REG (64)
+
SCAN_OUT
100
FUSE
COMPARE
102
–
PTU_ENABLE

FIG.3

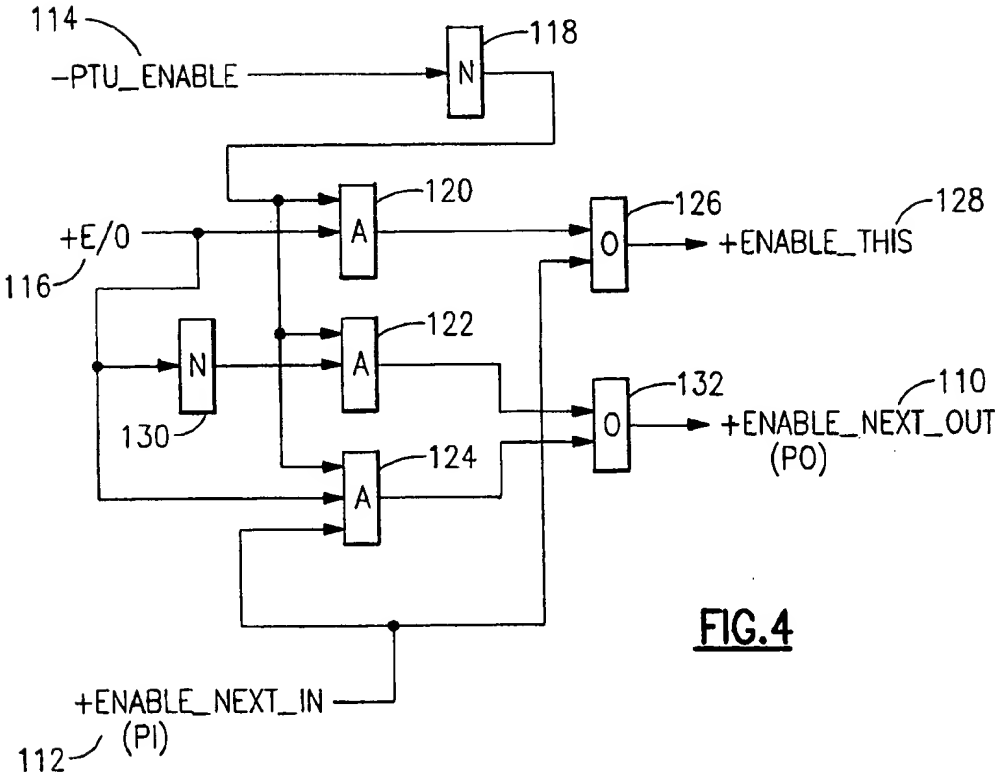
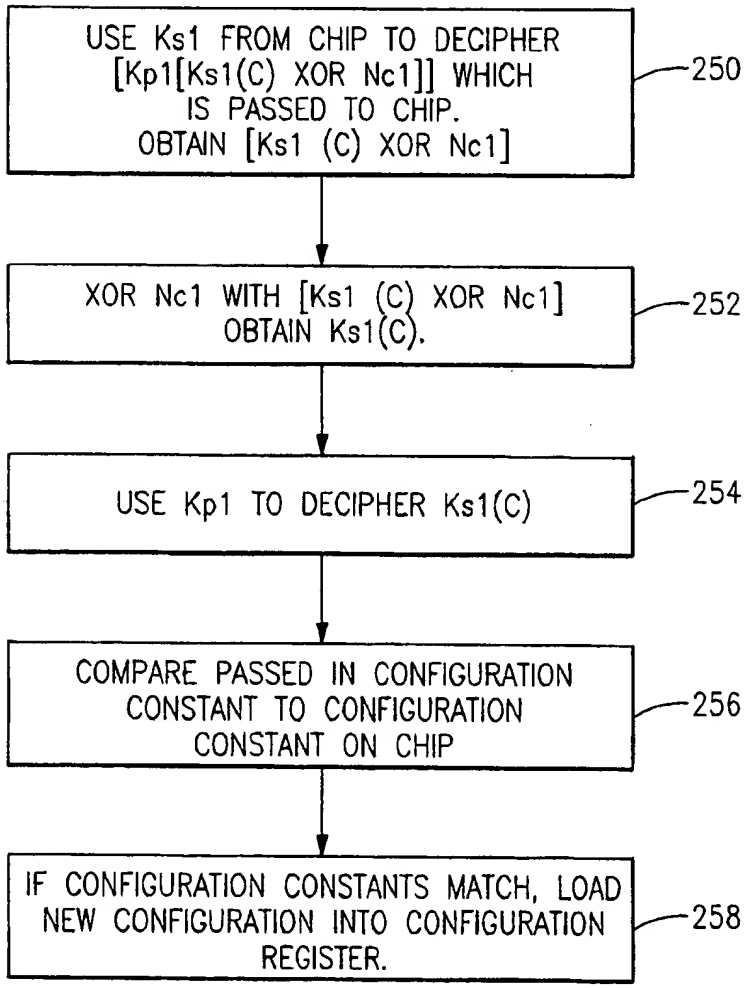

114
–PTU_ENABLE
N
118
120
A
+E/0
116
126
O
+ENABLE_THIS
128
N
122
A
130
132
O
110
+ENABLE_NEXT_OUT
(PO)
A
124
+ENABLE_NEXT_IN
(PI)
112

FIG.4

FIG.5

CONFIGURATION SIGNATURE	PUBLIC KEY	ID NUMBER
Ks1(C)	Kp1	ID1
Ks2(C)	Kp2	ID2
Ks3(C)	Kp3	ID3
Ks4(C)	Kp4	ID4
•	•	•
•	•	•
•	•	•

FIG.7



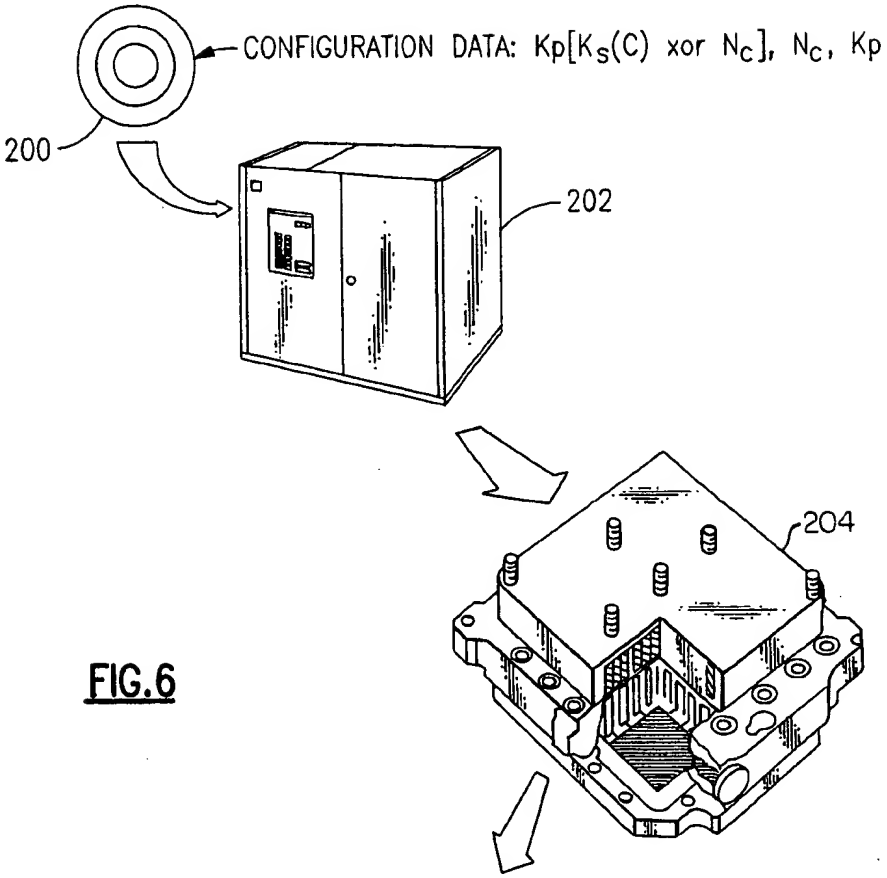
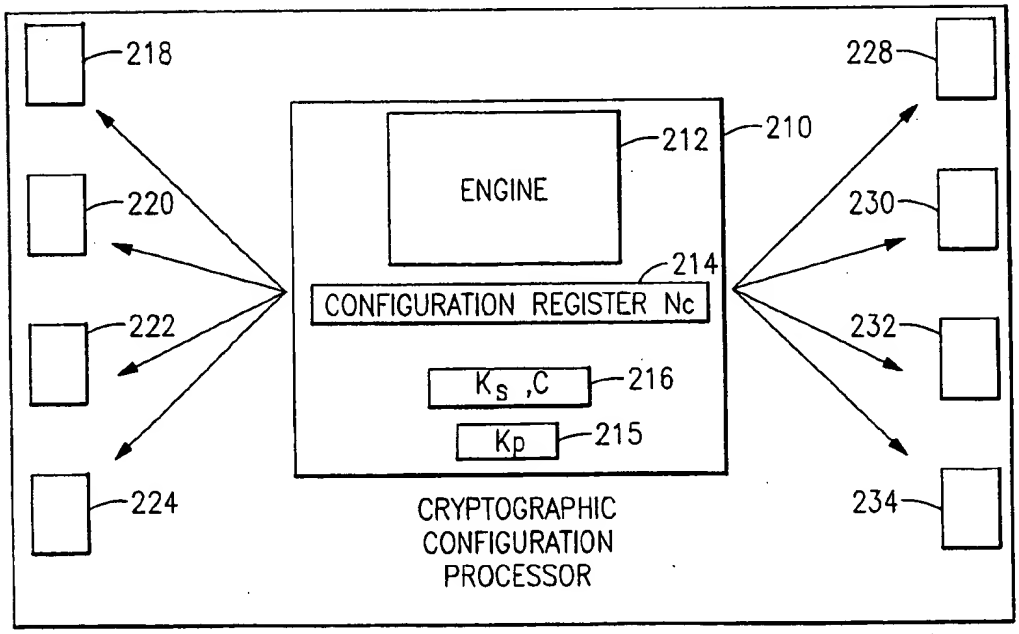

CONFIGURATION DATA: Kp[Ks(C) xor Nc], Nc, Kp
200
202
204

FIG.6


218
220
222
224
228
230
232
234
210
212
ENGINE
214
CONFIGURATION REGISTER Nc
Ks ,C
216
Kp
215
CRYPTOGRAPHIC CONFIGURATION PROCESSOR

METHODS AND APPARATUS FOR SECURE HARDWARE CONFIGURATION

BACKGROUND OF THE INVENTION

1. Field of the Invention

The present invention generally relates to methods and apparatus for electronically and securely configuring hardware features and options and, more particularly, to a configuration which contains a predetermined code or encryption sequence on a computer chip which is used to selectively enable the function features of the chip.

2. Description of Related Art

At present, it is standard for a variety of models of computers to be offered, each with different capabilities and features. This is true for each type of computer, from a personal computer to large mainframe and super computers. However, this wide variety of choices results in the requirement that each variation and model must be manufactured and kept in inventory.

In particular, it is common for a specific computer to be offered with various configurations of a multichip module or single chip assembly which each include different features and capabilities. Each module or chip assembly could include different numbers of control processors and feature chips. At present, each configuration of a module or chip assembly is manufactured and must be stocked in inventory to meet the demands of customers who desire the option of variations in functional characteristics or capabilities of a computer system. However, it is apparent that computer manufacturers would realize large efficiencies and economies by reducing the number of models manufactured.

In addition, a computer is generally sold with a particular configuration which meets the current needs of a computer user. The computer can usually be upgraded, modified or repaired as needed to meet the increased needs of the customer. At present, it is common for a new hardware unit to be added to the computer system to upgrade the hardware features or options. Often, a new or additional multichip module must be installed. The installation of the new hardware often must be performed off-site and so the customer must do without the computer system while the appropriate hardware is added or removed. In other instances, the configuration can be done on site but requires extensive configuration of the hardware. The loss of the use of the computer can result in financial loss and usually causes an interruption in the customer's business. Furthermore, similar problems result from failures in hardware elements.

In the past, off chip configurations have been used to store information for enabling features of a circuit. However, these systems are not completely secure since the information or code can be intercepted from the chip interface. In addition, a separate key chip can be used to provide security for single chip modules. The single chip modules are becoming more common as the density of a chip increases, so that multiple processors and/or features are present on a single chip.

Therefore, methods and apparatus which allow for the implementation of numerous combinations of functional characteristics or capabilities for a computer system or a multi chip module would be advantageous. It would also be beneficial to perform the configuration electronically and securely.

SUMMARY OF THE INVENTION

It is therefore an object of the present invention to provide methods and apparatus for secure electronic configuration of hardware in a computer system.

It is another object of the present invention to provide methods and apparatus which allow hardware features and options to be selectively enabled.

It is also an object of the present invention to provide methods and apparatus for dynamic feature or option upgradability.

It is a further object of the present invention to provide methods and apparatus for secure configuration of features and capabilities which are on the computer chip which is to be configured.

According to the invention, methods and apparatus are provided for electronically configuring hardware features and options. In particular, methods in a computer chip are encoded using a fusible array and/or encryption techniques In a first embodiment, computer chips are provided which use a set of interconnected wires with a pattern unique to each chip to represent a series of codes which can be used to selectively enable particular features and options of a computer system. The codes on the chip include a unique portion which is readable and a unqiue portion which is not readable. In a second embodiment, a unique encrypted key is burnt into a computer chip. This key is used in combination with configuration information to enable or modify the configuration of hardware features or options.

BRIEF DESCRIPTION OF THE DRAWINGS

The foregoing and other objects, aspects and advantages will be better understood from the following detailed description of the preferred embodiments of the invention with reference to the drawings, in which:

FIG. **1** is a block diagram showing a hardware configuration on which the subject invention may be implemented;

FIG. **2** is a diagram providing a high level functional overview of a representative multichip module (B) as shown in the central electronics complex of the computer system shown in FIG. **1**;

FIG. **3** illustrates the system of electronic hardware configuration according to a first embodiment of the invention;

FIG. **4** illustrates the configuration and system used to enable one or more additional computer chip if one of the enabled chips fails;

FIG. **5** is an example of a database configuration which correlates an identification number to a public key and a configuration signature for each chip which is manufactured;

FIG. **6** is a high level overview of electronic hardware configuration according to a second embodiment of the invention; and

FIG. **7** is a flow diagram of the method for deciphering a configuration constant which is passed to the chip to enable or disable a feature of the chip.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS OF THE INVENTION

Referring now to the drawings, and more particularly to FIG. **1**, there is shown a representative hardware environment on which the subject invention may be implemented. This hardware environment may be a personal computer such as the International Business Machines (IBM) Corpo-

ration's PS/2 Personal Computers; a workstation such as IBM's RS/6000 Workstations or a mainframe computer such as the IBM System/390.

As shown, the computer **10** is connected to an operator console **12** in a manner well known in the art. In addition, there can be included a main electronic storage **14** and a central electronics complex **16**. The central electronics complex can include one or more multichip modules or a single high density circuit with integrated circuit devices equivalent to several million transistors. A service processor **18** is preferably included between the operator console **12** and the central electronics complex **16** and provides access to the functions and circuitry within the complex. In addition, a power supply **20** and input/ output (I/O) channels **22** or adapters are generally provided as will be understood by those skilled in the art. The I/O channels are preferably connected to one or more Direct Access Storage Devices (DASDs), such as a diskette, tape storage devices, printers, terminals, disk drives or other similar devices.

The central electronics complex, as shown in FIG. **1**, generally includes four or more multichip modules. In the example shown in FIG. **1**, the central electronics complex includes an SC module **24** which buffers and controls the flow of data between the main store **14**, the input/output (I/O) module **26** and the processors in the computer or data processing system **10**. The input/output module **26** preferably controls and buffers data between I/O channels **22** and the main store **14**, as is well understood by those of skill in the art. The buffer (B) module **28** is preferably provided to buffer and control instructions and data used by the central processor (CP) module **30** executes instructions within computer **10**. As is well understood by those skilled in the art, each of the multichip modules, **24, 26, 28**, and **30** is a highly complex electronic module which can include greater than one hundred integrated circuit devices, each of which is equivalent to thousands or millions of transistors. It should also be understood that the multichip modules discussed and shown are merely representative of the type and number of modules which may be included.

FIG. **2** provides a high level schematic representation of one multichip module, the buffer (B) module, from the central electronics complex **16** of FIG. **1**. As shown, various control functions **40** are implemented by this multichip module. A buffer **42** is provided to buffer instructions and data from the CP module discussed above. In addition, generally a directory **46** and cache **48** are provided to buffer and control the data between the B module and the SC module. It is also generally understood that a translation lookaside buffer (TLB) **44** will be provided for translating virtual memory addresses into real memory addresses with the main store or other portions of the computer or data processing system. Therefore, it should be understood that the functional characteristics or features of the computer **10** can be modified and controlled by varying the capabilities and circuitry of a multichip module.

A customizable computer chip is used to selectively enable or configure features or functions of a multichip module The invention can also be used on single chip modules wherein more than one process is contained within a single chip. In a first embodiment, each of the features, processors or functions is associated with a fusible array on the chip. A fusible array is a set of wires which are connected to ground at one end and available to logic at the other end. The array of wires is customized by cutting open a predetermined combination of wires. Each chip will have a unique combination of wires or in other words, a unique code. The fusible array can have two distinct portions. The first, a readable portion, can be read by the system and is optional. The other portion is a secure and cannot be read by the system. Both portions are used to uniquely identify the chip. The readable unique portion and secure unique identification code are used to identify the configuration system and to modify or enable features or options of the system. A record is maintained of the information on the readable portion and this information can be used to determine the personality of the secure portion. It is also possible to use a single encryption for each chip, wherein the readable portion is encrypted to provide a second code. The second code corresponds to the secure portion. It is also possible for more than one array to be on a single chip so that multiple functions can be separately enabled.

It should be noted that the boundaries of the assembly, module and chip are artificial and that this invention can be used where such boundaries are sufficiently secure. As an example, more than one central processor could be packaged onto a single chip and each processor within a chip could have the secure enable function. Similarly, a central processor could include more than one chip and the secure enable function could be kept secure by packaging means where the secure enable function is within the same chip of the central processor chip set or where the chips are packaged within an assembly or module that is secure from tampering. It is possible to prevent tampering through the use of self destruct mechanisms, proprietary unique tools which are required to access the secure function or means that incorporate the off chip secure signals in an operational function such that the enable signal is a simple level but is part of a complex operation that could not easily be created off chip.

It is expected that the computer chip which is used to enable or determine the configuration will contain a **64** bit fuse macro. It is preferred that the computer chip include two **64** bit registers, one to encode the secure portion and one for the readable portion. As discussed above, the fuse macro is customized by cutting open combinations of wires.

As shown in FIG. **3**, fuse **100** represents the secure portion and fuse **106** represents the readable portion. Any conventional means can be used to read the readable portion, for example, a scan means can be used to load the contents of the readable portion into a scanable register. In the representative configuration shown, the register has two purposes. The first, is to read the readable unique fuse and second, and the second is to hold an entered key which must match the contents of the secure unique fuse **100** to enable one of the programmable functions on the chip.

In order to clarify the configuration, an example implementation is set forth. The readable fuse **106** is optional and contains a unique code which is read. The information obtained from the readable portion is used to cross reference, by look up table, enacryption or other means, a unique **64** bit code or key. The unique key is given to the purchaser of the chip(s) if the function(s) controlled by the secure portion are purchased. The key can be in an ASCII file, a hard copy hexadecimal number, on a floppy disk, on a CD-ROM or electronically transmitted to the system. It is understood that these are representative examples and other means could be used to supply the key. The key is loaded, by any means known in the art, into the register **108**. The key can be loaded during the power up routine or under system control. As described above, the comparator **102** compares the key to the secure unique code encoded on the secure fuse **100**. If the codes match then the corresponding function is enabled.

The use of a **64** bit register is preferred so that the possibility of guessing the unique key is reduced. It is

possible to use fewer bits in combination with other mechanisms to reduce the possibility of an unauthorized determination of the unique code. Some mechanisms might include a limit on the number of attempts or the performance of a main store operation between each attempt.

As a further example, a multichip module is provided, which can include up to ten control processors and five optional chip sites. The desired options or control processors are enabled if the key or secret code which matches the code on the control processor or chip is supplied. It is preferred that the keys or secret codes be supplied during power on reset. As discussed above, the keys can be read from a diskette or read only memory (ROM), supplied by the service processor or input using a keyboard or other input device. Each function, for example an option or control processor, has a unique code which will enable that particular function. As discussed in detail above, the unique codes are contained in fuses in a computer chip.

According to the example described above, a multichip module with ten control processors and five optional chip sites is manufactured. However, a customer only desires a system configured with four control processors and no optional chips. The keys for enabling this configuration are supplied with the multichip module to the customer so that the desired configuration can be enabled.

The methods and configurations described herein can also be used as an enable next daisy chain function, wherein a new chip is enabled if one becomes disabled. This allows a failing computer chip to be taken off line and replaced with a spare functional chip without knowing the secret code for the spare chip.

Each multichip module can have a plurality of daisy-chained functionally equivalent computer chips, for example control processors are daisy chained to other control processors or memory chips are daisy chained to other memory chips, any of which may be enabled provided its enabling code has been purchased or is otherwise known by the user. Any unused chips may be used as redundant chips and can be brought online should an enabled chip fail. In the preferred embodiment, the invention provides an ENABLE NEXT feature, wherein a failing chip is taken offline and replaced with an unused chip in a manner which is transparent to the user.

Referring now to FIG. **4**, there is shown a logic circuit for implementing the ENABLE NEXT feature. Each chip on the multichip module has associated therewith an ENABLE NEXT logic circuit daisy-chained to the ENABLE NEXT logic circuit of the next chip wherein an ENABLE_NEXT_OUT line **110** of a first chip is connected to the ENABLE_NEXT_IN line **112** of the next chip. Each chip in the chain has connected thereto an E/O latch line **116** which is held to logic 1 if the chip is functioning properly and to logic 0 if a chip failure is detected. If a chip is enabled and functioning properly, the PTU_ENABLE line **114** will be 0 and its complement **118** will be 1. The PTU_ENABLE complement **118** is input to AND gates **120**, **122** and **124**. The E/O line **116** is also connected to the input of AND gates **120** and **124**, and its complement **130** to AND gate **122**. When a chip is functioning properly and the E/O latch line is **1**, AND gate **120** **120** and OR gate **126** cause ENABLE_THIS line **128** to enable the chip. If the enabled chip fails or malfunctions and E/O becomes 0, ENABLE_THIS line **128** will take the chip offline, and OR gate **132** will cause ENABLE_NEXT_OUT line **110** to become active. Since the chips are daisy chained, the ENABLE_NEXT_IN line **112** of the next chip will be a **1** and cause the ENABLE_THIS line **128** of the next chip to enable that chip. In the event that the next chip is already enabled and functioning, AND gate **124** and OR **132** will activate the ENABLE_NEXT_OUT line **110** to the next chip and so on and so on until an unused chip is reached.

It is also possible to electronically configure hardware of a computer with an integrated cryptographic configuration processor computer chip. Most standard encryption algorithms can be used to encode the chip. However, it is preferred that a public key algorithm such as Rivest-Shamir-Adleman (RSA) is used and the example set forth below will refer to this method. However, it should be understood that these examples are not intended to be limiting and other techniques and public key encryption methods could be used within the scope of this invention.

Each cryptographic configuration processor chip has associated with it a unique RSA key pair, a secret key (Ks) and a public key (Kp), of which Ks is permanently coded or installed within the chip. An example of one technique which could be used to code the chip is laser delete fuses. In addition, a configuration constant (C) is also permanently coded or installed within the chip. Prior to encoding Ks into the chip, it is used to encrypt the value of the configuration constant to produce a configuration signature Ks(C). After the secret key is encoded on the chip and used to encrypt the configuration constant, Ks is preferably deleted or removed from all databases. A database, an example of which is shown in FIG. **5**, is maintained which correlates the serial number or other identification of each chip to the public key and the configuration signature (Ks(C)). Since Ks is unique for each chip, the configuration signature (Ks(C)) is also unique for each chip. The information stored in the database is used to securely enable the hardware configuration data.

Hardware can be electronically configured by sending a new configuration to the cryptographic configuration processor hardware with the appropriate unique configuration signature for the chip and the associated public key to sign the transaction. The signed configuration signature along with the new configuration data and the associated public key can be represented as follows:

Kp (Ks (C) xor Nc), Nc, Kp

In the above formula, Nc represents the new hardware configuration and the remaining variables are as discussed, supra. Although Nc is not a secret value, the configuration processor via the use of this encryption technique ensures that the sender of the configuration data is authorized to send such data.

As shown in FIG. **6**, the signed configuration signature, configuration data, and associated public key **200** are provided, preferably on an optical disk, to the multichip module **204** in a computer **202**. Each multichip module has at least one cryptographic configuration processor **210** which includes an engine **212**, a configuration register **214**, with each bit representing the various possible configurations, a public key register **215**, an embedded configuration constant, C, and an embedded secret key, Ks, **216**. This information is used to selectively enable and disable central processors and/or other functional chips **218**, **220**, **222**, **224**, **236**, **228**, **230**, **232**, **234**, **236** in the module. It should be understood that any number of central processors or chips can be included.

It is only possible for the chip to decipher the new configuration signature since the deciphering requires the secret key, Ks, which is embedded into the chip. After the configuration signature is deciphered, the value of Nc must

be unraveled from the signature via an exclusive or (XOR) function, as illustrated in FIG. 7. First, as shown in block **250**, the secret key Ks is used to decipher the configuration signature [Kp[Ks(C)⊕Nc]] which results in [Ks (C)⊕Nc], where ⊕ is the XOR function. Then, as shown in block **252**, Nc is XOR with [Ks(C)⊕Nc] to obtain Ks(C). Next, Ks(C) is deciphered using Kp in block **254**. The configuration constants are then compared in block **256** to verify the identity of the chip. If the configuration constants are identical, then the new configuration, Nc, is loaded into the configuration register in block **258**. The configuration signature for each chip is maintained in a database which is not be public and therefore unauthorized upgrades or modifications to the configuration are prevented.

This system requires only a single configuration signature for each computer since each new configuration can include many bits or values that reflect multiple upgrades or features for more than one chip or processor. Furthermore, since each computer or system has a unique configuration signature, if the configuration signature for a particular computer or system is compromised, only that computer or system can be altered or modified. Moreover, the use of a cryptographic algorithm reduces the probability that the contents will be unraveled. In addition, it is possible and preferred that the cryptographic configuration processor chip be physically designed to reduce the chance of determining the secret imbedded values (Ks).

While the invention has been described in terms of its preferred embodiments, those skilled in the art will recognize that the invention can be practiced with modification within the spirit and scope of the appended claims.

Having thus described our invention, what we claim as new and desire to secure by Letters Patent is as follows:

1. An electronic hardware configuration system, said system comprising:

a computer module in the form of a multichip module or an integrated circuit chip wherein said computer module includes a plurality of programmable features;

a first register on said computer module for storing a secure code unique to the computer module, said secure code including unique codes for each of said plurality of programmable features;

a second register for receiving a key code, said key code identifying a feature or features of said computer module to be enabled;

means connected to said first and second registers for comparing said key code to said secure code on said computer chip;

means for evaluating results from said means for comparing, wherein if said key code matches said secure code, a feature or features identified by said key code are enabled.

2. An electronic hardware configuration system, as recited in claim **1**, wherein the secure code and the key code comprise keys in a public-key cryptographic algorithm.

3. An electronic hardware configuration system, as recited in claim **2**, further including a customizable array of fuses, said array of fuses including said secure code encoded therein.

4. An electronic hardware configuration system, as recited in claim **3**, wherein said customizable array of fuses comprises fusible links.

5. An electronic hardware configuration system, as recited in claim **2**, wherein said secure code is stored in a programmable read only memory.

6. An electronic hardware configuration system, as recited in claim **2**, wherein said computer module comprises:

a plurality of computer chips;

means for connecting a first of said computer chips to a second of said computer chips if said first computer chip and said second computer chip have at least one programmable feature in common, wherein said first computer chip and the common programmable feature have been previously enabled but said second computer chip and the corresponding common programmable feature have not been enabled; and

means for enabling the second of said computer chips and the corresponding common programmable feature if the first of said computer chips fails.

7. An electronic hardware configuration system, as recited in claim **2**, wherein said computer module further stores a configuration constant and includes means for implementing said public-key cryptographic algorithm, said secure code and said configuration constant being encrypted in combination using said cryptographic algorithm to produce a configuration signature, said configuration signature being unique for each computer module.

8. An electronic hardware configuration system, as recited in claim **7**, wherein said key code includes a public key, a signed configuration signature and configuration data.

9. An electronic hardware configuration system, as recited in claim **8**, wherein said means for comparing of said key code includes means for deciphering said signed configuration signature so that said configuration constant stored in said computer module can be compared to said received and deciphered configuration constant.

10. An electronic hardware configuration system, as recited in claim **9**, wherein said received configuration data is used to identify which one of said programmable features is enabled.

11. An electronic hardware configuration system, as recited in claim **10**, wherein said computer module comprises:

a plurality of computer chips;

means for connecting a first of said computer chips to a second of said computer chips if said first computer chip and said second computer chip have at least one programmable feature in common, wherein said first computer chip and the common programmable feature have been previously enabled but said second computer chip and the corresponding common programmable feature have not been enabled; and

means for enabling the second of said computer chips and the corresponding common programmable feature if the first of said computer chips fails.

12. An electronic hardware configuration system, as recited in claim **1**, wherein said computer module comprises:

a plurality of computer chips;

means for connecting a first of said computer chips to a second of said computer chips if said first computer chip and said second computer chip have at least one programmable feature in common, wherein said first computer chip and the common programmable feature have been previously enabled but said second computer chip and the corresponding common programmable feature have not been enabled; and

means for enabling the second of said computer chips and the corresponding common programmable feature if the first of said computer chips fails.

* * * * *

US005774544A

United States Patent [19]

Lee et al.

[11] Patent Number: 5,774,544

[45] Date of Patent: Jun. 30, 1998

[54] **METHOD AN APPARATUS FOR ENCRYPTING AND DECRYPTING MICROPROCESSOR SERIAL NUMBERS**

[75] Inventors: **Sherman Lee**, Rancho Palos Verdes, Calif.; **James R. MacDonald**, Buda; **Michael T. Wisor**, Austin, both of Tex.

[73] Assignee: **Advanced Micro Devices, Inc.**, Sunnyvale, Calif.

[21] Appl. No.: **623,024**

[22] Filed: **Mar. 28, 1996**

[51] **Int. Cl.**[6] **H04L 9/00**

[52] **U.S. Cl.** **380/4**; 380/23; 380/25; 380/49; 380/50; 380/9

[58] **Field of Search** 380/4, 9, 23, 25, 380/28, 49, 50, 59

[56] **References Cited**

U.S. PATENT DOCUMENTS

5,029,207	7/1991	Gammie	380/23 X
5,237,610	8/1993	Gammie et al.	380/23 X
5,319,705	6/1994	Halter et al. .	
5,602,920	2/1997	Bestler et al.	380/49

FOREIGN PATENT DOCUMENTS

0 707 270	4/1996	European Pat. Off. .	

OTHER PUBLICATIONS

Dallas Semiconductor Corp.: "Section 1: Introduction," Oct. 6, 1993, Data Book Soft Microcontroller, pp. 1–3, 07/08, 73, 77–80, 82, 229, 290–292.

Electronics, "Designer's Dream Machine," vol. 60., No. 5, Mar. 1987, New York, US, pp. 53–57.

Ferreira, R.C.: "The Smart Card: A High Security Tool in EDP,"Philips Telecommunication Review, vol. 47, No. 3, Sep. 1, 1989, pp. 1–19.

International Search Report for PCT/US 97/05117 dated 08–04–97.

AMD5$_K$Processor Technical Reference Manual, 1986 Advanced Micro Devices, Inc., pp. 3–29 through 3–31.

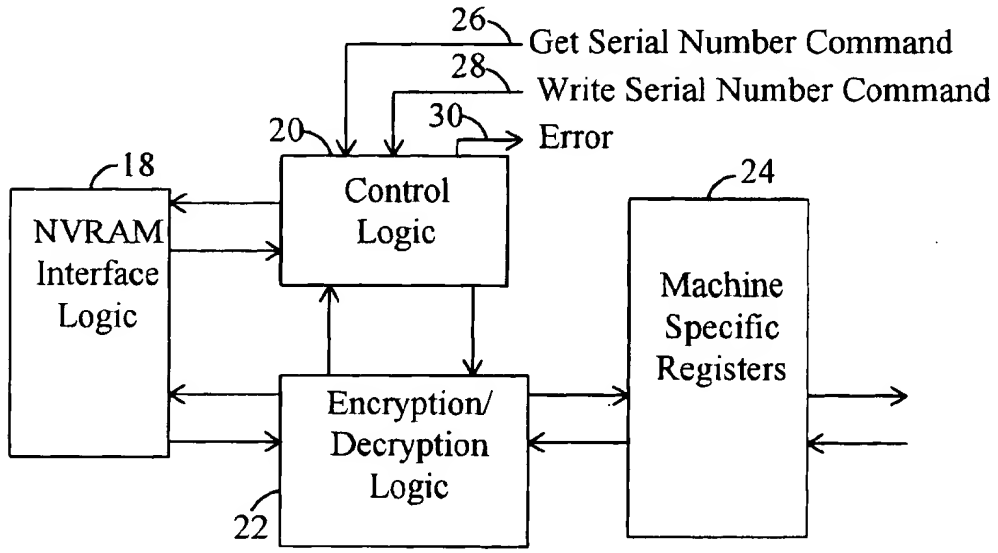
Primary Examiner—Bernarr E. Gregory

Attorney, Agent, or Firm—Conely, Rose & Tayon; B. Noel Kivlin

[57] **ABSTRACT**

A method and apparatus for encrypting and decrypting a microprocessor serial number. First and second encryption keys and a serial number are provided in microprocessor machine specific registers. The serial number is encrypted using the first key. The encrypted serial number is encrypted using the second key. The first encryption key may be encrypted along with the serial number using the second key. The double encrypted serial number is then stored in memory provided for that purpose.

13 Claims, 8 Drawing Sheets



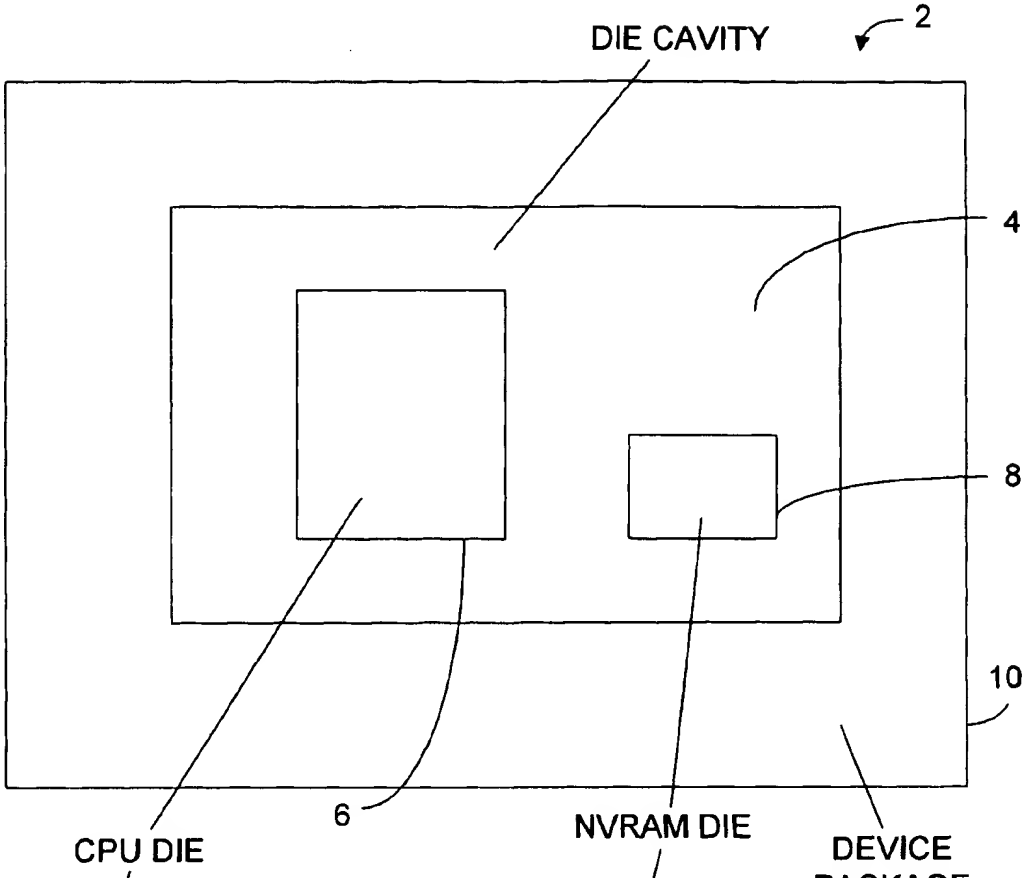

DIE CAVITY
2
4
8
10
6
CPU DIE
NVRAM DIE
DEVICE PACKAGE

Fig. 1a

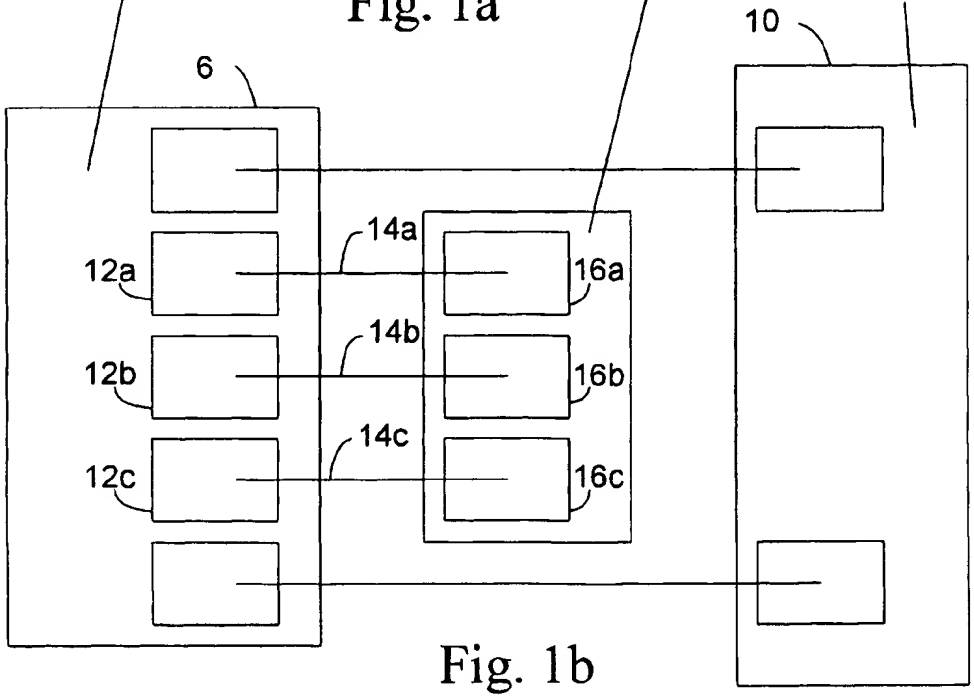

6
10
12a
12b
12c
14a
14b
14c
16a
16b
16c

Fig. 1b

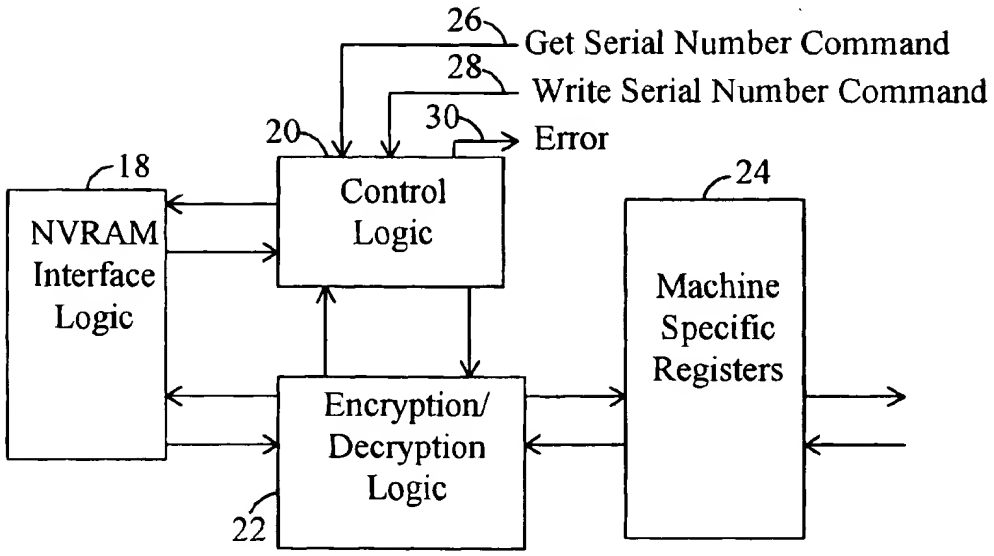

26
Get Serial Number Command
28
Write Serial Number Command
30
Error
20
18
NVRAM Interface Logic
Control Logic
24
Machine Specific Registers
Encryption/ Decryption Logic
22

Fig. 2

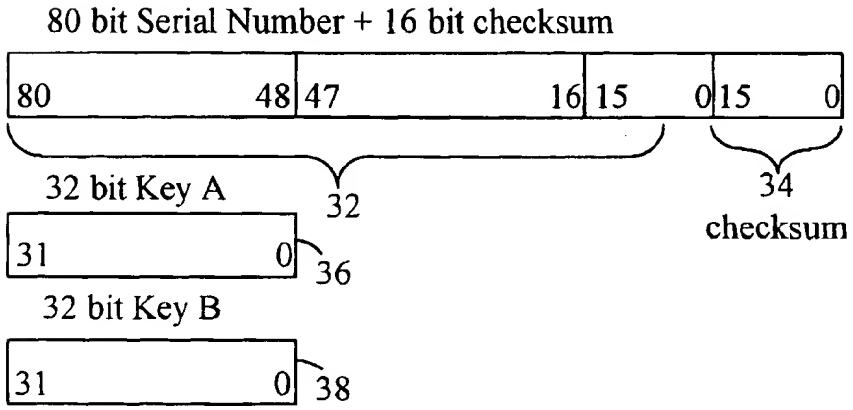

80 bit Serial Number + 16 bit checksum
80
48
47
16
15
0
15
0
32 bit Key A
32
34
checksum
31
0
36
32 bit Key B
31
0
38

Fig. 3

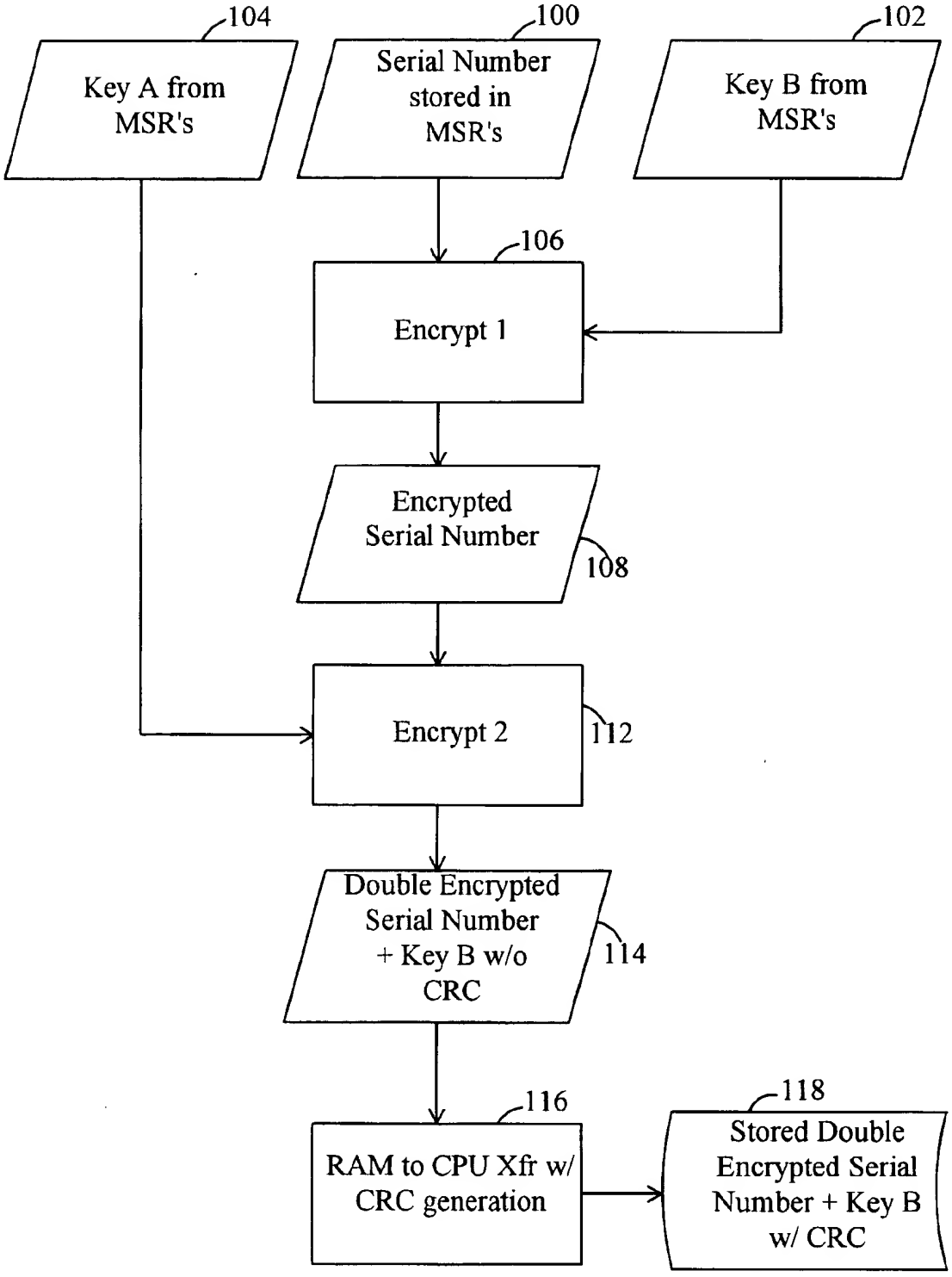

104
Key A from MSR's
100
Serial Number stored in MSR's
102
Key B from MSR's
106
Encrypt 1
Encrypted Serial Number
108
Encrypt 2
112
Double Encrypted Serial Number + Key B w/o CRC
114
116
RAM to CPU Xfr w/ CRC generation
118
Stored Double Encrypted Serial Number + Key B w/ CRC

Fig. 4a

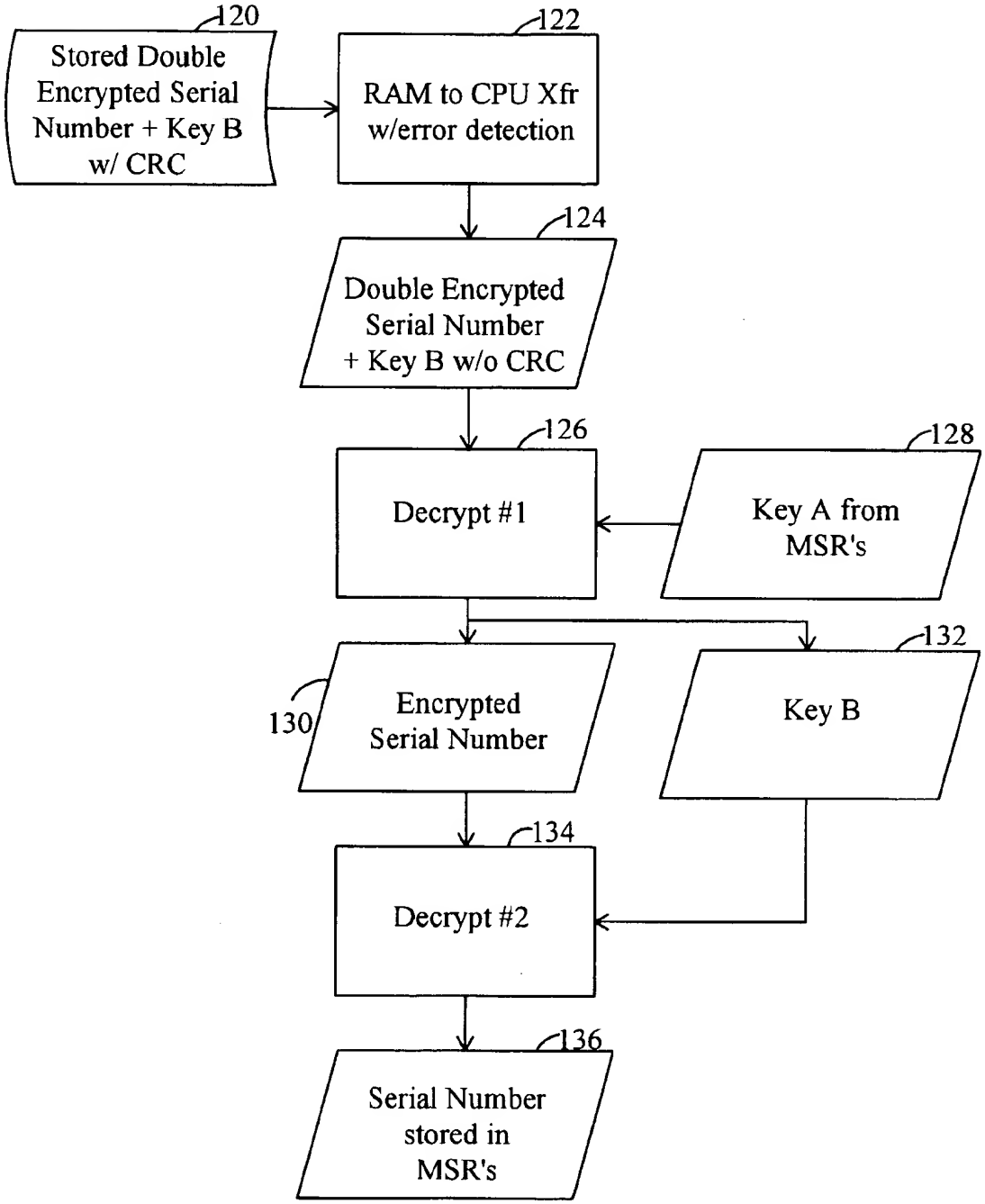

120
Stored Double Encrypted Serial Number + Key B w/ CRC
122
RAM to CPU Xfr w/error detection
124
Double Encrypted Serial Number + Key B w/o CRC
126
Decrypt #1
128
Key A from MSR's
130
Encrypted Serial Number
132
Key B
134
Decrypt #2
136
Serial Number stored in MSR's

Fig. 4b

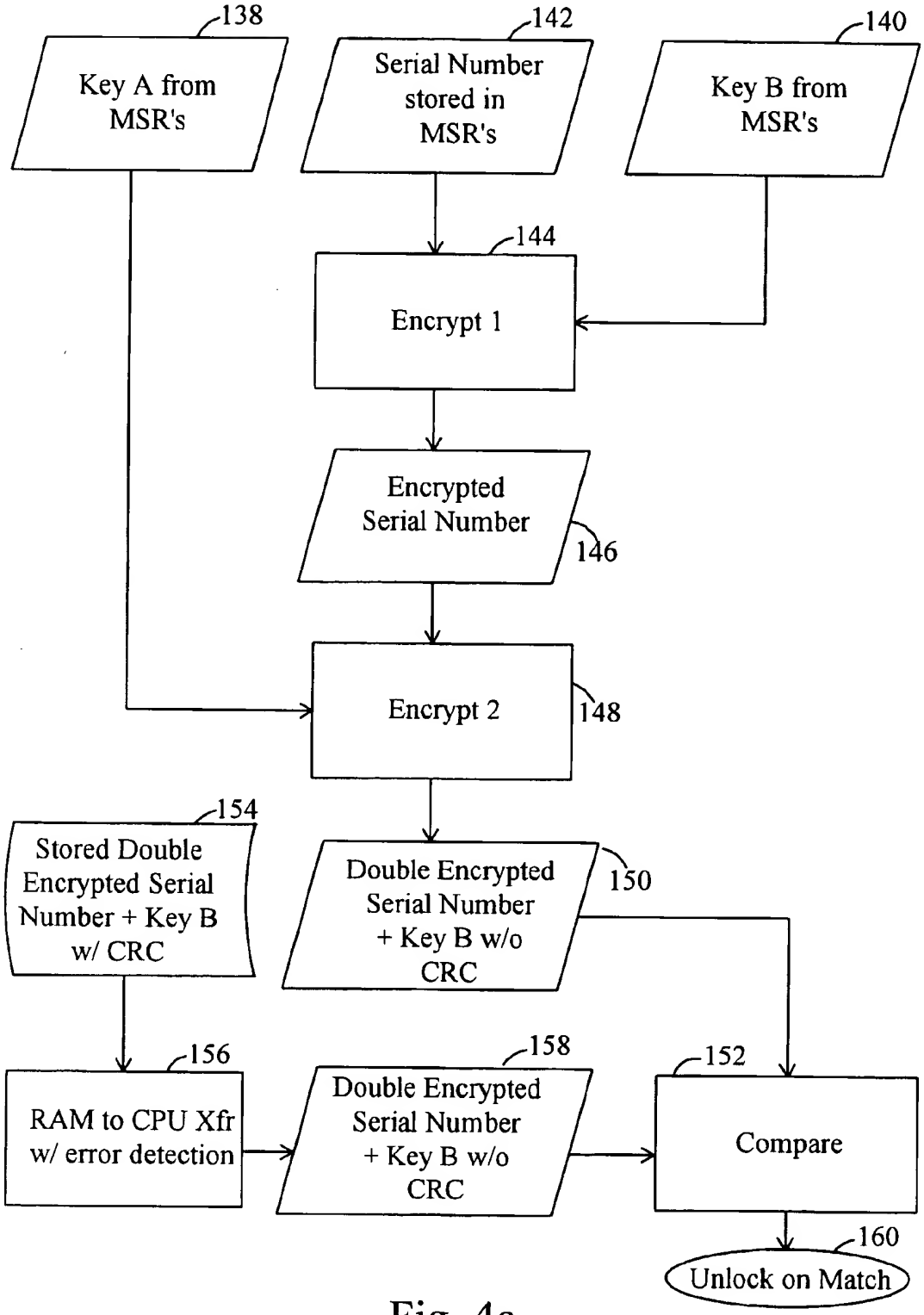

138
Key A from MSR's
142
Serial Number stored in MSR's
140
Key B from MSR's
144
Encrypt 1
Encrypted Serial Number
146
Encrypt 2
148
154
Stored Double Encrypted Serial Number + Key B w/ CRC
Double Encrypted Serial Number + Key B w/o CRC
150
156
RAM to CPU Xfr w/ error detection
158
Double Encrypted Serial Number + Key B w/o CRC
152
Compare
160
Unlock on Match

Fig. 4c

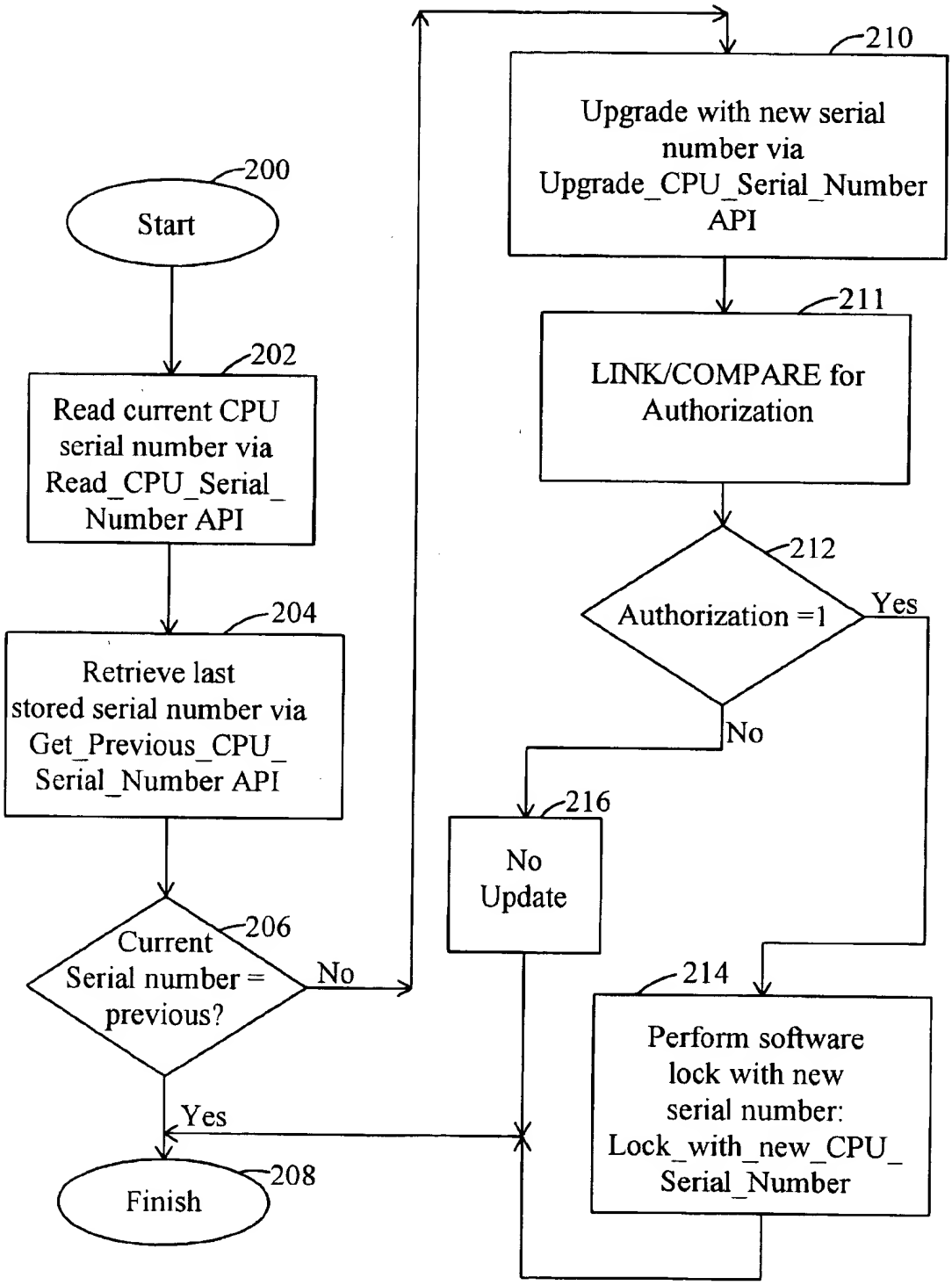

200
Start
202
Read current CPU serial number via Read_CPU_Serial_Number API
204
Retrieve last stored serial number via Get_Previous_CPU_Serial_Number API
206
Current Serial number = previous?
No
Yes
208
Finish
210
Upgrade with new serial number via Upgrade_CPU_Serial_Number API
211
LINK/COMPARE for Authorization
212
Authorization =1
Yes
No
216
No Update
214
Perform software lock with new serial number: Lock_with_new_CPU_Serial_Number

Fig. 5

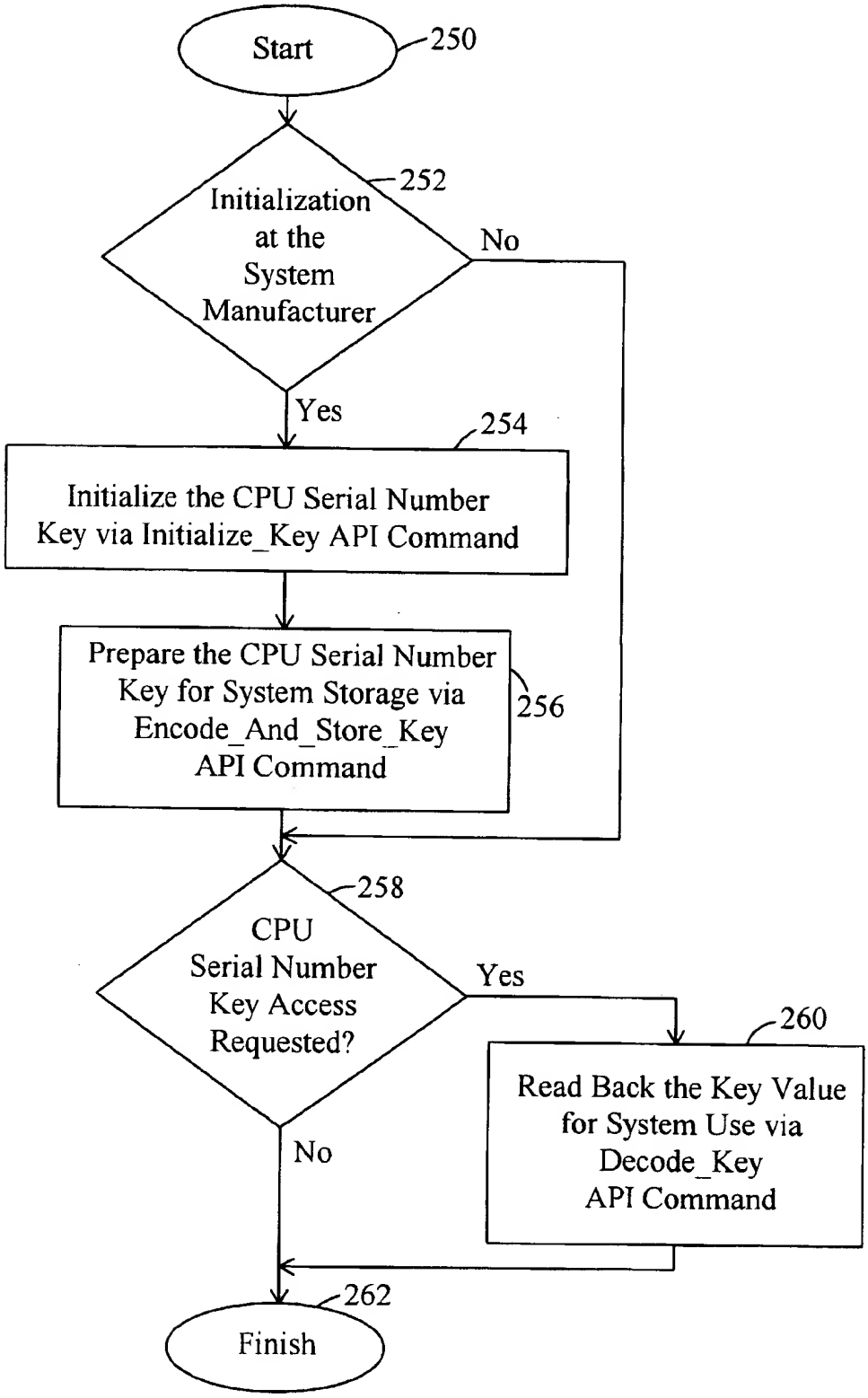

Start
250
Initialization at the System Manufacturer
252
No
Yes
254
Initialize the CPU Serial Number Key via Initialize_Key API Command
Prepare the CPU Serial Number Key for System Storage via Encode_And_Store_Key API Command
256
CPU Serial Number Key Access Requested?
258
Yes
260
Read Back the Key Value for System Use via Decode_Key API Command
No
262
Finish

Fig. 6

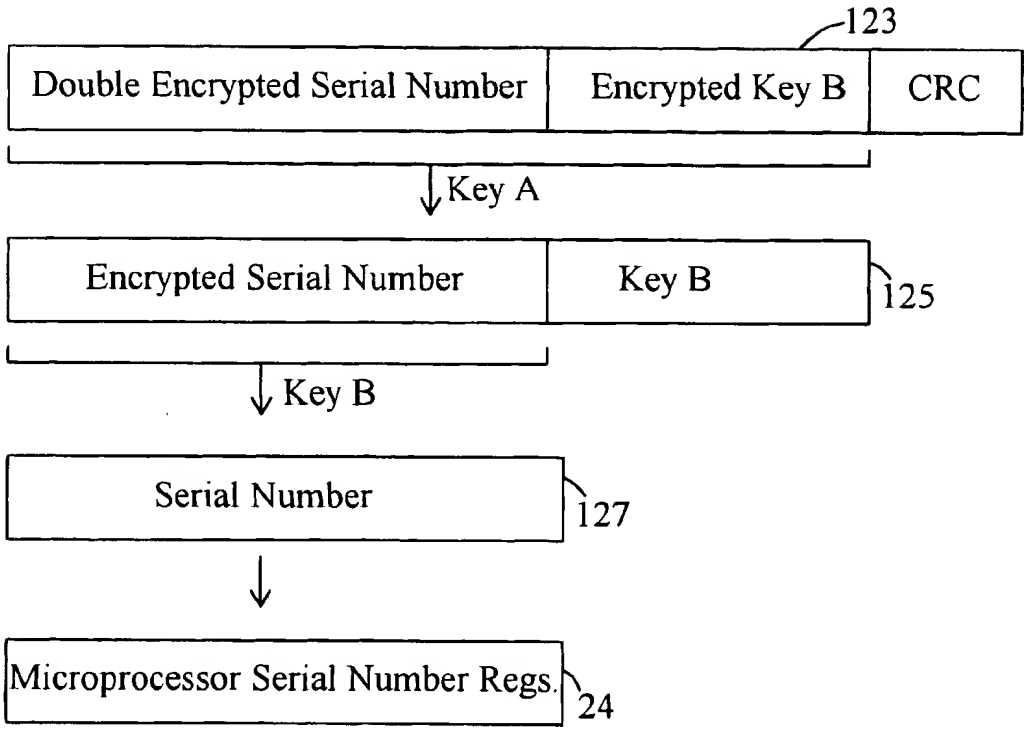

123
Double Encrypted Serial Number
Encrypted Key B
CRC
Key A
Encrypted Serial Number
Key B
125
Key B
Serial Number
127
Microprocessor Serial Number Regs.
24

Fig. 7

METHOD AN APPARATUS FOR ENCRYPTING AND DECRYPTING MICROPROCESSOR SERIAL NUMBERS

FIELD OF THE INVENTION

The present invention relates to encrypting and decrypting microprocessor serial numbers and particularly to a system and method for providing two layers of encryption to microprocessor serial numbers.

Description of the Related Art

For some time, workstations, minicomputers, and mainframes have had serial numbers embedded in them which uniquely identify them. Addition of a serial number allows a manufacturer to trace a product in the field back to the original equipment manufacturer (OEM). This allows the manufacturer greater control over its product. In addition, provision of a serial number permits independent software vendors to register their products. Microprocessors and personal computers typically have not been tracked by serial number, partly due to the added expense of providing circuitry to store and/or read a serial number.

As the complexity of the microprocessors themselves has increased, however, it is becoming increasingly cost-effective to provide additional circuitry and/or process steps to provide a serial number. Moreover, a serial number may be associated with particular software. Thus, for example, software that is installed on a particular processor may read a password and thereafter be keyed to the particular microprocessor wherein an attempt to install the software on a different processor would fail. Nevertheless, standard methods of providing serial number identification in a computer system are generally expensive; in a competitive microprocessor market, it is desirable to keep such costs at a minimum. Accordingly, there is a need for an inexpensive, yet effective way of providing a serial number with a microprocessor in order to trace a product in the field back to the original equipment manufacturer (OEM).

A problem with providing a microprocessor serial number in machine readable form is that it can become accessible to unauthorized users and thus susceptible to unauthorized alteration. Accordingly, there is a need for a mechanism to prevent unauthorized access to a machine readable serial number.

Still another problem with providing a microprocessor serial number and serializing the software is that if the processor is upgraded or otherwise replaced, the software will cease to function. (Serializing software is herein defined as providing a CPU serial number to a given set of software. The software will be not able to run on a processor not having that serial number.) However, to the software, there is little difference between being loaded onto an unauthorized computer system and having an unauthorized processor provided to it. In either case, the software will be keyed to a processor that is no longer present and will not function. Accordingly, what is needed is an upgrade method whereby serialized software can detect that it is running on an unauthorized processor and in response thereto can initiate a reauthorization process. If the reauthorization process is successful, the software will function on the upgrade processor. Failure at the reauthorization process, however, will mean that the software itself is loaded onto an unauthorized system and, hence, not function.

As discussed above, it is also desirable to serialize software. To do so, however, it is desirable to provide an easy method of accessing the serial number by the software while

at the same tine maintaining the serial number's inaccessibility to unauthorized changes.

SUMMARY OF THE INVENTION

Accordingly, there is provided a unique system and method for providing, maintaining and upgrading the software lock of a microprocessor. A mechanism is provided for storing a microprocessor serial number in a nonvolatile random access memory formed within the same device package as the processor. The microprocessor serial number is encrypted using a double-key encryption scheme in order to prevent unauthorized access and alteration. An encryption key is itself encoded to provide easy access to an authorized user, while preventing unauthorized reading of the serial number. Finally, there is provided a method whereby software that has been serialized to a particular processor can detect that it is running on an unauthorized processor and request reauthorization.

The present invention thereby prevents unauthorized access to or alteration of a microprocessor serial number. Thus, manufacturers can maintain greater control over their products.

Accordingly, there is provided a small, nonvolatile random access memory packaged with the CPU die to provide a storage space for the CPU serial number which can be programmed before leaving the factory. Both the CPU die and the nonvolatile RAM die reside within the cavity of the package. Connection between the two die is provided by conventional wire bonding and kept to a minimum by providing a serial interface between the RAM and the CPU.

In accordance with another aspect of the present invention, access to the nonvolatile RAM storing the CPU serial number is controlled by encryption and logic on the processor. Two small layered encryption keys are used to increase security of the mechanism. The serial number may be changed only if both keys are correct. The keys and encryption algorithm are known only to the manufacturer. Register space is provided for an 80-bit serial number and two 32-bit keys. A checksum is included in the data stored in the RAM to allow detection of errors in the transfer of the RAM data to and from the CPU.

In accordance with yet another aspect of the present invention, an upgrade method is provided whereby serialized software detects that it is running on an unauthorized processor and initiates a reauthorization process based on a reauthorization use profile. The temporary reenabling of the software is allowed if the authorization service is not available or not allowed. Limited use is provided to the user until the problem is resolved.

In accordance with still another aspect of the claimed invention, a code sequence is provided for detecting a serialized CPU, extracting the CPU serial number and providing it to applications by a standard application program interface (API).

Broadly speaking, the invention contemplates a method employing a first encryption key and a second encryption key to provide a first layer of encryption and a second layer of encryption to a microprocessor serial number. The resulting double-encrypted serial number is then stored in memory set aside for that purpose.

BRIEF DESCRIPTION OF THE DRAWING

A better understanding of the present invention can be obtained when the following detailed description of the preferred embodiment is considered in conjunction with the following drawings, in which:

FIG. 1*a* and FIG. 1*b* are block diagrams showing a nonvolatile RAM die and CPU die sharing a device package in accordance with one embodiment of the claimed invention.

FIG. 2 is a block diagram of one embodiment of an encryption system in accordance with one embodiment of the present invention.

FIG. 3 is a diagram of a register set encryption system in accordance with one embodiment of the present invention.

FIGS. 4*a*, 4*ba* and 4*c* are flowcharts illustrating the write-read and unlock processes of encrypting in accordance with one embodiment of the present invention.

FIG. 5 is a flowchart illustrating an upgrade reauthorization system in accordance with one embodiment of the present invention.

FIG. 6 is a flowchart illustrating accessing the serial number is accordance with one embodiment of the present invention.

FIG. 7 is a diagram illustrating encryption in accordance with one embodiment of the present invention.

While the invention is susceptible to various modifications and alternative forms, specific embodiments thereof are shown by way of example in the drawings and will herein be described in detail. It should be understood, however, that the drawings and detailed description thereto are not intended to limit the invention to the particular form disclosed, but on the contrary, the intention is to cover all modifications, equivalents, and alternatives following within the spirit and scope of the present invention, as defined by the appended claims.

DETAILED DESCRIPTION OF THE INVENTION

Turning now to FIG. 1*a*, a diagram is shown of an integrated circuit package **2** which includes central processing unit (CPU) die **6**, nonvolatile random access memory (NVRAM) die **8**, die cavity **4**, and device package **10**. In a preferred embodiment, NVRAM die **8** is formed separately from CPU die **6**. Turning now to Figure 1*b*, NVRAM die **8** is coupled to CPU die **6** by means of signal lines **14***a*, **14***b*, **14***c*. Signal lines **14***a*, **14***b*, and **14***c* are attached to CPU die **6** and NVRAM die **8** by means of bond pads, **12***a*, **12***b*, **12***c* and **16***a*, **16***b*, **16***c*, respectively. In one embodiment, signal line **14***a* is a transmit line; signal line **14***b* is a receive line; and signal line **14***c* is a clock line, and form a serial interface between the NVRAM and the CPU. In an alternative embodiment, NVRAM die **8** may be connected to CPU die **6** by means of a single transmission line. In that embodiment, the CPU and NVRAM include additional circuitry to provide the serial interface. The NVRAM is programmed with the CPU serial number prior to leaving the factory.

While providing the CPU serial number in NVRAM packaged with the CPU die provides cost advantages over, for example, permanently etching a serial number onto the die, a mechanism is needed to prevent unauthorized access to the serial number. Access to the NVRAM and hence the serial number, is controlled via encryption keys and logic on the processor. Turning now to FIG. 2, there is shown a block diagram of an exemplary encryption mechanism. The CPU includes NVRAM interface logic **18**, coupled to control logic **20** and encryption/decryption logic **22**. Control logic **20** and encryption/decryption logic **22** are further coupled to one another. Control logic **20** supervises the read, write serial number processes. Encryption/decryption logic **22** performs the encryption and decryption of the serial number as described below. Encryption/decryption logic **22** is also coupled to machine or model specific registers **24**. Machine specific registers **24** provide the programmers interface and are of a type common in advanced x386-type processors and include command and/or status bits (e.g., unlock, read, etc.). It should be noted, however, that the use of other processors or registers is contemplated. Control logic **20** is further coupled to Get Serial Number command line **26**, Write Serial Number command line **28**, and Error line **30**. NVRAM interface logic **18** controls transfers to and from the NVRAM.

FIG. 3 illustrates 80-bit serial number **32**, 32-bit key A **36**, and 32-bit key B **38**. Eighty bit serial number **32**, and keys **36**, **38** are stored in machine specific registers **24**. Sixteen bit checksum **34** is further included in the data stored in the NVRAM to allow detection of errors in the transfer of the data to and from the CPU. However, the checksum **34** is not used in the encryption process. A layered encryption method using the two keys **36**, **38** is employed to increase the security of the mechanism. The serial numb er itself may be changed only if both keys **36**, **38** are correct. The keys and the encryption algorithm are known only to the manufacturer.

Turning now to FIG. 4*a*, a flow diagram of a write process used to write a new serial number is shown. Initially, serial numbers are provided in the machine specific registers (MSR) **24** (step **100**), as are keys **36**, **38** (step **104**, step **102**). Next, in step **106**, key **38** is used to provide a first level of encryption to the serial number stored in the machine specific registers **24**. The encrypted serial number (step **108**) and key B are then further encrypted using key A in step **112**. The now double-encrypted serial number are shown in step **114**, with cyclical redundancy checksum (CRC). Next, in step **116**, the double encrypted serial number with encrypted key B is output to the NVRAM using CRC generation. In step **118**, the resulting encrypted serial number with CRC is stored.

Turning now to FIG. 4*b*, a flow diagram of a read process is shown. The stored double-encrypted serial number and encrypted key B with CRC is initially stored (step **25** **120**) in the NVRAM. In step **122**, an NVRAM to CPU transfer occurs using error detection. The resulting double-encrypted serial number and encrypted key B without CRC (step **124**) is then decrypted in step **126** using key A from the machine specific registers (step **128**). Having decrypted using key A in step **130**, what remains is the encrypted serial number with key B (step **132**). Key B is then used to decrypt the encrypted serial number in step **134**. The resulting completely decpted serial number is then stored in the machine service registers in step **136**. A schematic representation of the decryption process may be found in FIG. 7. Key A is applied to double-encrypted serial number and encrypted key B **123**. The resulting encrypted serial number **125** is then decrypted using key B. The resulting serial number **27** is then stored in microprocessor serial number (or machine specific) register **24**.

The above described read and write processes are permitted only in the event that the CPU is in an unlocked state. This can occur upon programming of a new serial number for the first time. For example, when the processor is first assembled, the NVRAM is zeroed out. The processor will detect this state and enter the unlocked state which will allow the initial serial number to be programmed by the write method described above. If, however, the device previously had a serial number within it, the unlock sequence described below must be run in order to reprogram

the serial number. More particularly, turning now to FIG. **4c**, in an initial state, the serial number is stored in the machine specific registers, as are keys A, B (steps **138**, **140**, and **142**). The serial number is then encrypted using key B in step **144**. The resulting encrypted serial number (step **146**) and key B are further encrypted in step **148** using key A. The resulting double encrypted serial number and encrypted key B in step **150** are then input into a comparator in step **152**. At the same time, the previously stored double-encrypted serial number with encrypted key B and CRC (step **154**) is transferred from NVRAM to the CPU with the appropriate error detection in step **156**. In step **158**, the double-encrypted serial number and key B is output to the comparator (step **152**). The two outputs from steps **150** and **158** are compared in comparator **152**. If there is a match, the processor will enter an unlock state (step **160**).

As can readily be appreciated, the processes described above are necessarily processor specific. Increasingly, however, easy upgrades of processors are available. A processor upgrade will cause software that is linked to the processor's serial number ("serialized software") to cease functioning. In accordance with one aspect of the claimed invention, a method is provided whereby serialized software can detect that it is running on an upgraded (unauthorized) processor and will initiate a reauthorization process. In a preferred embodiment, the reauthorization procedure is carried out through use of a series of application programming interfaces (API). The relevant APIs are set forth below:

Read_CPU_Serial_Number
This function allows the OS and application calling it to enable the reading
of the CPU serial number. In one embodiment, in order to read the serial number a 32-bit key must be entered as well as setting a read_serial_number bit in the appropriate MSR.
Entry:
MSR to be accessed for the serial number
The 32 bit key
Exit:
CPU Serial Number
Upgrade_CPU_Serial_Number
This function will perform the automatic upgrade and re-authorization process when a CPU upgrade has occurred.
Entry:
New serial number
Old serial number
Exit:
Authorization == 0 - not allowed
Authorization == 1 - allow upgrade
Lock_With_New_CPU_Serial_Number
If the Upgrade_CPU returns an Authorization = 1 then the corresponding application will take the appropriate actions to change the software locking
scheme utilized.
Entry: New serial number
Exit: Success == 1 -> lock with new serial number OK
Success == 0 -> lock with new serial number FAILED
Get_Previous_CPU_Serial Number
This function provides a mechanism for the OS and application to retrieve what the previous CPU serial number stored into the system.
Entry:
nothing
Exit:
Success == 1 or 0
if Success == 1
Previous CPU serial number stored by the system
else
nothing

The authorization process is detailed in FIG. **5**. The procedure is initiated in step **200**. At step **202**, the current CPU serial number of the installed microprocessor is read via the Read_CPU_Serial_Number API command described above. In the next step, step **204**, the most recently stored serial number is retrieved via the Get_Previous_CPU_Serial Number API command. If the current CPU serial number is the same as the previous CPU serial number (step **206**), then the process is finished in step **208**. If, however, the two are not equal, then in step **210**, the Upgrade_CPU_Serial_Number API command will be initiated and authorization sought. Authorization is obtained by contacting the vendor via a telephone voice, data, Internet connection **211** or other remote connection. Based on an authorization use profile, the reenabling of the software may be allowed in step **212**. If authorization is permitted, then in step **214**, the software lock will be performed using the new CPU serial number and the Lock_with_new_CPU_serial_number API command (step **214**). If authorization is not allowed, then in step **216**, the upgrade will not be permitted. In either case, the process is completed in step **208**. If the authorization service routine is not available, or authorization is denied, then one embodiment contemplates an API for allowing limited use so that the user may use this system until the problem is resolved.

One potential gap in the encryption mechanism described above is that the 32-bit key A and 32-bit key B are stored in the machine specific registers. It should be noted that, although both keys are necessary to write the serial number, only one need be provided to read it. Accordingly, the key value that is needed to read the serial number must be encoded in order to protect against users retrieving the key value. In addition, it is desirable to reduce the data size of the key from four bytes to one or two bytes in order to preserve CMOS. This aspect of the claimed invention is best described in reference to FIG. **6** as well as the APIs described below:

Initialize_Key
This function can only be initiated by the systems manufacturer. The purpose of this function is to facilitate the storage of the 32-bit key value into the CMOS. This function will encode the key and store into CMOS.
Entry:
32-bit Key value
CMOS index for storage of the encoded value
Exit:
Nothing
Encode_And_Store_Key
This function will be called by the Initialize_Key function. This function will encode the key and store it in the system CMOS. The key will not be 100% protected but will prevent the normal users from decoding the key information from system CMOS memory.
Entry:
32-bit Key value
CMOS index for storage of the encoded value
Exit:
Nothing
Decode_Key
This function will retrieve the encoded key value from CMOS and decode it for use
Entry:
CMOS index for stored encoded key value
Exit:
32-bit Key value
The following diagram illustrates the system software access and control of the serial number and key.

More particularly, with reference to FIG. **6**, coding the serial number key begins at step **250**. If the systems manufacture has initialized the key (step **252**), then the Initialize_Key API command will be asserted (step **254**), in order to initialize the serial number key. Upon initialization, the CPU serial number key is prepared for system storage (step **256**) via the Encode_and_Store_Key API command. Among other things, the Encode_and_Store_Key API command encode the key. For example, the function may provide a

summation of the key bytes and store them in system CMOS or a more complex encoding. If key access is not requested (step **258**), then the process is completed (step **252**). If, however, the key number access is requested, then the key value is read back for system use (step **260**) using the Decode_Key API command. Once the key value has been retrieved, reading the serial number may proceed as described above. While the above system and method will not provide complete protection against unauthorized access to the key or serial number, the casual user will not be able to gain unauthorized access.

The invention described in the above-detailed description is not intended to be limited to the specific form set forth herein, but on the contrary, it is intended to cover such alternatives, modifications, and equivalents as can reasonably be included within the spirit and scope of the invention as defined by the appended claims.

We claim:

1. (amended) A method of associating a serial number with a microprocessor, comprising:

writing a first encryption key, a second encryption key, and a serial number to a register in a microprocessor;

accessing said register for said first encryption key, said second encryption key, and said serial number;

encrypting said serial number using said first encryption key to obtain an encrypted serial number;

encrypting said encrypted serial number using said second encryption key to obtain a double-encrypted serial number; and

storing said double-encrypted serial number in a nonvolatile random access memory.

2. (amended) The method of claim **1** wherein said storing step includes generating and storing a cyclical redundancy checksum.

3. A computer system for encrypting a microprocessor serial number comprising:

a central processing unit including at least one register for storing at least one encryption key and an unencrypted microprocessor serial number;

a memory unit coupled to said central processing unit for storing an encrypted microprocessor serial number;

encryption/decryption circuitry coupled to said at least one register and said memory unit for encrypting said unencrypted microprocessor serial number using said at least one encryption key; and

interface circuitry coupled to said encryption/decryption circuitry and said memory unit for transferring said encrypted microprocessor serial number to said memory unit.

4. (amended) The computer system of claim **3** wherein said memory unit is a nonvolatile random access memory.

5. (amended) The computer system of claim **3**, wherein said at least one register stores a first encryption key and a second encryption key.

6. (amended) The computer system of claim **3** wherein said encryption/decryption circuitry performs a first layer of encryption and performs a second layer of encryption using said first encryption key and said second encryption key.

7. (amended) The computer system of claim **3** wherein said at least one register is a machine specific register.

8. (amended) A method of unlocking access to a microprocessor serial number, comprising:

writing a first encryption key, a second encryption key, and a first serial number to a register in a microprocessor;

writing a second, double-encrypted serial number in a memory unit accessible to said microprocessor;

accessing said register for said first encryption key, said second encryption key, and said first serial number;

encrypting said first serial number using said first encryption key to obtain an encrypted serial number;

encrypting said encrypted serial number using said second encryption key to obtain a second double-encrypted serial number;

comparing said first double-encrypted serial number and said second double-encrypted serial number; and

entering an unlocked state wherein the microprocessor serial number is accessible if said first double-encrypted serial number and said second double encrypted serial number are the same.

9. A method for enabling a software program to execute on only a single microprocessor, wherein the method comprises:

storing a double-encrypted serial number with an encrypted first key in a processor's nonvolatile memory;

storing an encoded second key in a system-level nonvolatile memory; and

configuring the software program to:

perform a first API call to decode and retrieve the second key;

perform a second API call with the second key and an address of a machinespecific serial number register to decode the double-encrypted serial number to retrieve a decrypted serial number;

compare the decrypted serial number to a previously stored serial number; and

proceed with execution if the decrypted serial number matches the previously stored serial number.

10. The method of claim **9**, wherein the configuring step further comprises configuring the software program to:

request authorization if the decrypted serial number does not match the previously stored serial number; and

store the decrypted serial number as the previously stored serial number and proceed with execution if authorization is received.

11. The method of claim **9**, wherein the second API call comprises:

writing the second key to a second machine specific register;

transferring the double-encrypted serial number with the encrypted first key from the processor's nonvolatile memory to a third machine specific register;

decrypting the double-encrypted serial number and the encrypted first key using the second key to produce an encrypted serial number and a first key; and

decrypting the encrypted serial number using the first key to produce the decrypted serial number.

12. The method of claim **9**, wherein the step of storing the double-encrypted serial number with the encrypted first key comprises:

entering an unlocked state;

writing a first key to a first machine specific register;

writing the second key to a second machine specific register;

writing a serial number to a third machine specific register; encrypting

the serial number using the first key and concatenating the first key to produce an encrypted serial number;

encrypting the encrypted serial number using the second key to produce the double encrypted serial number which includes the encrypted first key;

generating a redundancy checksum for the double-encrypted serial number;

appending the redundancy checksum to the double-encrypted serial number;

transferring the double-encrypted serial number with the redundancy checksum to the processor's nonvolatile memory.

13. The method of claim **12**, wherein the step of entering an unlocked state comprises:

writing the first key to the first machine specific register;

writing the second key to the second machine specific register;

writing the serial number to the third machine specific register;

encrypting the serial number using the first key and concatenating the first key to produce an encrypted serial number;

encrypting the encrypted serial number using the second key to produce a new double encrypted serial number;

retrieving the double-encrypted serial number with the redundancy checksum from the processor's nonvolatile memory;

determining that the retrieved double-encrypted serial number matches with the new double-encrypted serial number.

* * * * *